University of Glasgow | School of Computing Science

Interactive Presenter - Mobile Collaboration Application

Ben Callis

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — March 22, 2013

Abstract

This project focuses on creating an iOS application, Interactive Presenter, which can be used to present documents to others in an engaging way. Presenters can use their iOS device to control a presentation to an audience of other iOS devices via Wi-Fi or Bluetooth, without any additional hardware. During a presentation, presenter's interactions are monitored and automatically distributed to the audience to keep the presentation synchronised across devices. Presenters can also send out questions, to assess the audiences understanding. Results are automatically collated on the presenters device and they can choose whether to share the answer and results with the audience. Viewers can also raise questions through the application without disrupting the presenter or revealing their identity.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____ Signature: _____

Contents

Chapter 1

Introduction

1.1 Motivation

Presentations are often delivered using slides displayed with a computer and digital projector. However, existing computer-based presentation systems limit flexibility in delivery, requiring a certain room layout with the presenter situated towards the front. This form of presentation has limited interaction, where audience members are passive recipients of information. Presenters often attempt to increase interaction by asking questions to gain feedback on how well the audience understand the presented material. This approach is severely limited in presentations with a large audience, as only a few selected viewers will be given the chance to answer questions. Interactivity can also occur in the form of spontaneously asked questions by members of the audience. However, not everyone is willing to ask questions due to social fear or time constraints.

In recent years, electronic voting systems have been employed as a means to increase interactivity during presentations. These systems utilise wireless technology to allow members of the audience to respond anonymously to questions posed by the presenter. Audience responses can be visually displayed during the presentation, to give immediate feedback to the presenter and the audience. However, these systems are rarely deployed due to increased hardware configuration and cost.

Audience members are increasingly likely to carry mobile devices with large graphical displays and networking capabilities. These devices could be used to deliver presentations in any room configuration without the need for a projector, whilst increasing interactivity by providing mechanisms found in electronic voting systems. Furthermore, this system would ensure that all members of the audience have the same viewing experience, irrespective of their position within the room.

1.2 Background

Electronic Voting Systems
Several studies have been performed to assess the benefits of using interactive voting systems in academia. In 2011, interactive handsets were trialled in several lectures at the University of Glasgow[13]. Results showed that EVS handsets were generally well received by students and lectures. One lectured commented *"I found the handsets very beneficial in my lecture and speaking with some students afterwards they also appreciated it"*. During the trial, 61 philosophy students were asked to rate the usefulness of the EVS, 77% of students rated the system between useful - extremely useful. These students identified anonymity as a key benefit *"The anonymity allows students to answer without embarrassing themselves"*.
Additional research by Simpson and Olivier, concluded that voting systems are best understood as a tool rather than a teaching approach, which when used in lectures can support increased motivation and attainment [15].

Mobile Devices
One of the ways in which mobile devices are often used together is for wireless multiplayer gaming. For example, iOS devices[1] support a large catalogue of multiplayer games, which users can play locally across devices with minimal configuration. There are several toolkits available to aid the development of networked games. However, this technology has not been appropriated for use in productivity environments.

There are two major operating systems that run on mobile devices: AndroidAndroid is a Linux-based operating system designed primarily for touchscreen mobile devices and iOS. Android has a consumer market share of 70% [12]; however, in enterprise the iOS platform dominates due to increased security. A recent report from Citrix, revealed that 58% of devices enrolled in its enterprise mobility services were running iOS[7]. Cook revealed that iOS devices are deployed in 94% of companies in the Fortune 500 [2] [9]. As presentations are common in corporate environments, this project produces a solution for the iOS platform. An overview of the iOS platform and development tools is included in appendix B.

1.3 Aim

The aim of this project is to develop a tool that allows presentations to be conducted in an interactive way using mobile devices, without any additional hardware or configuration. The application should:

- Allow a presenter to control a presentation from a single device.
- Support presentation viewing on multiple devices.
- Allow presenters to interact with the audience by distributing questions.
- Allow members of the audience to interact with the presenter by raising questions.

1.4 Outline

The remainder of this document is divided into the following chapters:

Chapter 2 - **Requirements**
This chapter discusses the projects requirements and the variety of ways in which they were elicited.

Chapter 3 - **Design**
This Chapter describes the visual and architectural design of the application.

Chapter 4 - **Implementation**
This chapter describes the implementation of the project in detail and describes user feedback obtained during development.

Chapter 5 - **Testing**
This chapter covers evaluations carried out on the project.

Chapter 6 - **Conclusion**
This chapter summarises the report and describes ideas for future development.

[1] mobile devices designed and marketed by Apple Inc. that run a Unix-like operating system named iOS. `http://www.apple.com/uk/ios/`

[2] The 500 largest corporations in the United States., compiled by Fortune magazine

Chapter 2

Requirements

This chapter discusses the projects requirements and the variety of ways in which they were elicited.

2.1 Questionnaire

To gain user feedback towards existing voting handsets, an online questionnaire was created. The questionnaire included specific questions targeted at participants who had used voting systems when presenting, and questions for participants who had used handsets when viewing a presentation. The majority of the questions were closed format, with the exception of a few which allowed for elaboration. There were 34 participants in total. 23 of these participants were currently in full-time education and 11 were in current employment. A summary of the results is included below.

Over half of the participants owned at least one iOS device.
For question 1, *Which of the following devices do you own?*, 55% stated that they had an iOS device. This suggests that developing the application for the iOS platform will make it accessible to a wide audience.

The majority of participants who had used voting handsets whilst presenting felt it increased engagement.
During question 5, *Have you ever used interactive voting handsets when presenting?*, eight participants stated they had used voting handsets whilst presenting. 70% of these participants said the audience appeared more engaged due to the handsets.

Two different voting handsets were identified
Presenters were asked to state which voting handsets they had used. The most popular handset identified was *ActiVote*[1], followed by *SMART Response*[2]. These two handsets are discussed in detail in section 2.2.1.

Presenters who had used voting handsets stated question set-up time and cost could be improved
For question 11, *What do you think could be improved?*, the majority of presenters (71%) stated that question set-up time and cost could be improved in future systems. Based on this, quick question set-up time was identified as a non-functional requirement. The project should also reduce costs as it makes use of existing hardware.

Hardware set-up complexity can prevent voting handsets being used.
During question 11, *What do you think could be improved?*, one participant stated *"Handsets are difficult to use when lessons are frequently held in different rooms due to the set-up time. Often have to get the IT Admins to*

[1] ActiVote - `http://www.prometheanworld.com/us/english/education/products/assessment-and-student-response/activote/`

[2] SMART Response - `http://smarttech.com/Solutions/Education+Solutions/Products+for+education/Complementary+hardware+products/SMART+Response/SMART+Response+PE`

install software onto computers before lesson. Which in many cases is not feasible". This indicates that voting handsets may be used more frequently if they did not need to be configured.

Viewers sometimes see voting handsets as a toy or game.
One participant commented *"I feel that the handsets are sometimes seen as a toy or a gimmick. They do have real value but it can be lost on some pupils who just want a 'game' instead of a lesson"*. To overcome this, the application should hide the voting controls until required.

The majority of participants who had used voting handsets whilst presenting felt it increased engagement.
Twenty-four participants stated they had used voting handsets when viewing a presentation. During question 13, *Do you feel using the interactive handset helped you to become more engaged in the presentation?*, 90% of these users stated the handsets increased their engagement. One user stated it had no effect on their engagement and another user said it decreased their engagement. These results are similar to findings by Jim Boyle (Professor at Strathclyde University). Boyle revealed that students generally regard interactive equipment as an advantage "70% for it, 20% indifferent, 10% definitely opposed to it" [10]. This indicates that the application could be of benefit to students and professionals.

Almost all participants stated they would be willing to use a voting system as a viewer or presenter
For question 14, *Would you be open to using a voting system as a viewer in a presentation?*, the majority (93%) answered yes. The same participants also stated they would be willing to use a voting system when presenting. These results are significantly higher than the number of participants who have used voting handsets during a presentation, which suggests that the unavailability of the voting handsets is limiting their use.

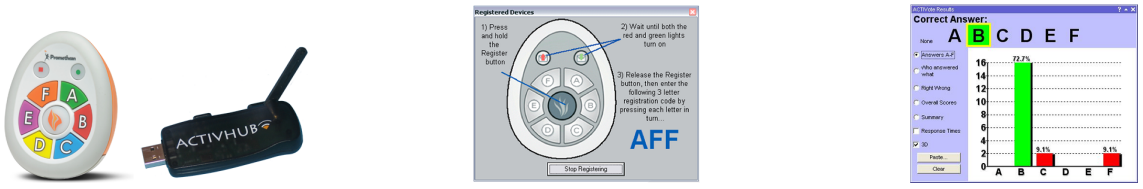
The full set of questions and results are included in appendix E.

2.2 Existing Products

2.2.1 Voting Handsets

Following on from the questionnaire, research was carried out on two voting handsets identified by presenters.

ActiVote
ActiVote is a student response system that enables teachers to poll students at any time during class to assess progress and, based on responses, tailor lessons to specific student needs. The system compromises of three parts: voting handsets, networking receiver and computer software. These components are illustrated in figure 2.1. Before the system can be used, software and device drivers need to be installed on a computer. Once installed, each device must be manually paired with the networking receiver (illustrated in figure 2.1b). The software allows multiple choice questions to be created and integrated with PowerPoint slide sets. Each question can support up to six possible answers. Viewers can answer questions by selecting one of the choice buttons labeled A-F. However, the handsets provide almost no feedback when answering a question, as they lack a physical display. The system is sold with 32 handsets for approximately £600.



(a) Handset and receiver (b) Handset pairing (c) Results screen

Figure 2.1: ActiVote student response system

SMART Response
The SMART Response interactive response system combines handheld wireless handsets, a receiver and pow-

erful assessment software (illustrated in figure 2.2). Users are required to install software and pair the handsets before using the system. The handsets include a small screen to allow users to view their response, and a telephone style keypad to allow textual responses. The system supports six question types: true or false, yes or no, multiple choice, multiple answer, numeric and text response. The software includes a 92 page user guide which describes how to configure and use the system. The software and 32 handsets cost approximately £1000.

(a) Handset and receiver (b) Handset buttons (c) Results screen

Figure 2.2: SMART Response voting handsets

2.2.2 Existing mobile applications

There are a few applications in the iOS AppStore that allow collaboration across devices. However, no applications exist which allow users to present information and interact with users through the use of questions. Three existing applications were analysed, to help capture requirements. User feedback towards each of these applications was assessed to determine what users want and expect from applications in this category.

(a) Conference Pad (b) Idea Flight (c) ResponseWare

Figure 2.3: Three collaboration applications currently available in the AppStore.

Conference Pad[3] allows PDFs to be wirelessly presented to up to fifteen iPads, iPhones, and iPod Touches. The user interface for the application is extremely simple consisting of just two screens: a presentation list and a document viewer. The document viewer allows the presenter to change page, zoom and use a laser pointer. These actions are instantly replicated on all connected devices. Users are required to load PDFs[4] onto their device using proprietary PC software.
Feedback is generally positive towards this application; however, the following issues were identified from user reviews in the AppStore:

- Importing PDF documents into the application from a computer is limited and time consuming.
- The application has no help section or user guide.
- All viewers need to purchase the application. Feedback suggests that their should be a free 'viewer' version.
- No way to unlock the presentation to allow viewers to navigate the PDF at their own speed.
- Viewers have no way to save the presentation.

Idea Flight[5] is similar to Conference Pad, in that it allows PDFs to be wirelessly presented to other devices, although it only supports iPads. The application offers the following additional features:

[3]Conference Pad - `http://www.regularrateandrhythm.com/apps/conference-pad/`
[4]Portable Document Format (PDF) is a file format used to represent documents
[5]Idea Flight - `http://www.ideaflight.com`

- Presenters can unlock the presentation to allow viewers to navigate the slides at their own pace.
- Presenters can annotate slides.
- Linkedin[6] integration - allowing the presenter to get to know their audience
- Note taking - allowing viewers to take textual notes during a presentation.
- Multiple import options - DropBox and 'Open in' support

Although this application is more feature-rich when compared to Conference Pad, it does not include the laser pointer functionality. It was also found to be fairly slow during testing on the original iPad. User feedback indicates that users are happy with the features on offer. However, there are several negative reviews relating to the cost and reliability of the application.

ResponseWare is a web-based service that allows questions to be created and responses to be obtained through smartphones or any web-enabled device. Voting via ResponseWare is an alternative to using traditional voting handsets. To use the mobile application, users need to pay a subscription fee and sign into the application. Once signed in, users can view current polls and submit answers to the questions. Results are presented to voters visually using graphs, making them easy to interpret. The service provides support for multiple-choice and open-ended questions; however, they must be created using the web based interface, which increases set-up time.

2.3 Functional Requirements

From the various requirement capture activities, a list of functional requirements was formalised. Functional requirements capture the intended behaviour of the system. These requirements were structured in terms of priority using the MoSCoW method[7], which splits requirements into four priorities: 'must have', 'should have', 'could have' and 'would be nice to have'. Requirements are unordered within each priority, as it is likely that multiple requirements have the same importance.

The following requirements represent functionality the application **must have** to be considered a success.

FR	Requirement	Description
1	View Documents	Allow users to view and navigate through documents
2	Wirelessly send Documents	Users must be able send a document wirelessly to other users running the application.
3	Synchronised page switching	Presenter page changes should automatically be replicated on viewers' devices.
4	Ask multiple choice questions	Presenters must be able to send multiple choice questions to viewers. Viewers must be able to select and send an answer to the presenter. Presenters must be able to share the answer or results with viewers.

The following requirements represent functionally the application **should have**, if possible. These items have high-priority and should be included in the final product.

FR	Requirement	Description
5	Laser pointer support	Presenter should be able to point to parts of the document using a laser pointer. This should show on viewer's devices
6	Zoom support	Presenters should be able to zoom in, to enlarge content on the page. This should automatically propagate to viewer's devices.
7	Cloud storage integration	Users should be able to import documents into the application from online cloud storage services.

[6] Linkedin is a social networking website for people in professional occupations - `http://www.linkedin.com`
[7] MoSCow prioritisation method - `http://www.coleyconsulting.co.uk/moscow.htm`

The following requirements are considered desirable but not necessary. They **could be** included in the final application if feasible within the time allocated to development.

FR	Requirement	Description
8	Navigation lock\unlock	Presenters should be able to toggle lock\unlock a presentation, so viewers can navigate slides at their own pace.
9	Save documents from a presentation	Allow the presenter to specify if viewers can store the document at the end of presentation.
10	Ask viewers open-ended questions	Allow presenters to send out open-ended questions to viewers during a presentation. Viewers should be able to submit a textual answer. Presenters would have the option to select the top answers and send them out to all viewers.
11	'Open in' support	Documents should be able to be brought into the application from other applications on the device.
12	Import documents from a PC	Allow documents to be imported into the application from a computer.
13	Note taking via keyboard	Allow viewers to make textual notes when viewing a presentation.
14	Passcode Secure presentations	Allow presenters to add passcodes to presentations to ensure only specific viewers can join the presentation.
15	Ask the presenter questions	Allow viewers to ask the presenter questions during a presentation.
16	Document syncing	Documents should be synced across all of the user's iOS devices.

This final set of requirements are unlikely to be included in the final implementation. They are considered as **would be nice to have** requirements and may be added in future releases.

FR	Requirement	Description
17	Presenter document annotation	Allow the presenter to annotate documents during a presentation, the annotations would appear on all devices viewing the presentation.
18	Viewer document annotation	Allow the viewers to directly annotate the document. These annotations would be saved locally with the document (providing the presenter granted the viewer permision to save the document) and could be looked at later.

As over half of the functional requirements were identified with a 'Could have' or 'Would be nice to have' priority, a user feedback and refinement evaluation was scheduled to take place after the major functionality had been implemented (discussed in section 4.2). This evaluation was performed to capture user feedback and further prioritise the requirements during development, to ensure any remaining development time was spent on user requested features.

2.4 Non-functional Requirements

Non-functional requirements describe how well a system performs its goals. They are fundamentally different to functional requirements, which describe the intended behaviour of the system.

NFR	Requirement	Description
1	Run on iPhone & iPad	The application should also be universal to cater for differing screen sizes.
2	Support iOS 5 +	Recent statistics from AudioBooks; a popular iOS app developed by David Smith show that 86% of devices using the application run iOS 6, 12% run iOS 5 and only 2% run iOS 4 [2]. To attract a large potential audience and use recent APIs the application should support iOS 5+.
3	Application should be under 50MB	Apple imposes a 50MB over-the-air download limit for the AppStore. The application size should be within this limit to allow the application to be downloaded without the need for a Wi-Fi connection.
4	Presentation privacy controls	The application should allow presenters to secure their presentations to control audience access. Presenters should also be able to specify if a presentation can be saved.
5	Viewer privacy	When viewers ask questions they should have the ability to remain anonymous.
6	Conserve battery life	Processor intensive tasks should be kept to a minimum to conserve the devices battery as much as possible.
7	Allow for future expansion	The application should be built to allow for additional features to be added.
8	Orientation support	The application should support both orientations.
9	Quick to add questions	Users should be able to add questions to a document within seconds of launching the application.
10	Automatic configuration	The application should automatically discover live presentations with no advanced set-up.

Chapter 3

Design

This chapter discusses the design of the application, presenting an overview of the overall system structure, as well as an in-depth account of the user interface design decisions.

3.1 User Experience

A series of storyboards were created to illustrate how the application will satisfy the key functional requirements. The first four storyboards focus on the functionality accessible to the two key actors of the system: a presenter or a viewer. The remaining three storyboards have a greater level of detail, illustrating the application in real-word scenarios. Each segment of a storyboard focuses on one primary task which usually directly maps to one screen of the application, allowing the flow of the application to be determined. The storyboards were roughly sketched with pen and paper and presented to potential users to explain how the application will work. Each storyboard is briefly described below stating the requirements it satisfies and a list of screens required is formalised.

3.1.1 Preparing for a Presentation

Figure 3.1 illustrates the steps required in preparing for a presentation. Before giving a presentation, slides need to be loaded into the application. On a traditional computer, slides are often created directly within a presentation program or downloaded from an external source. The application stays true to this well known workflow, allowing slides to be imported from multiple external sources including: other applications on the device (FR11), online file storage (FR7) and through desktop computers via iTunes (FR12). Once the slides are imported, presenters can rehearse their presentations (FR1) and add questions prior to presenting to others (FR4, FR11). To ensure the audience has the application installed, they can be notified ahead of time by sending a notification the application. From this storyboard the following screens were identified:

- Document list
- Document viewer
- Question list
- Add question
- Invite Friends

3.1.2 Presenting a document

Figure 3.2 demonstrates the application flow when presenting a document. To present, a document first needs to be imported into the application (explained above in section 3.1.1). Once imported, a user can open the document (FR1) and start the broadcast by tapping the broadcast button (FR2). This allows other devices to find and join the presentation. The user can then present their document to others, knowing the audience's devices are all in sync (FR3). During the presentation, the presenter can zoom (FR6), point (FR5), annotate (FR17) and send out

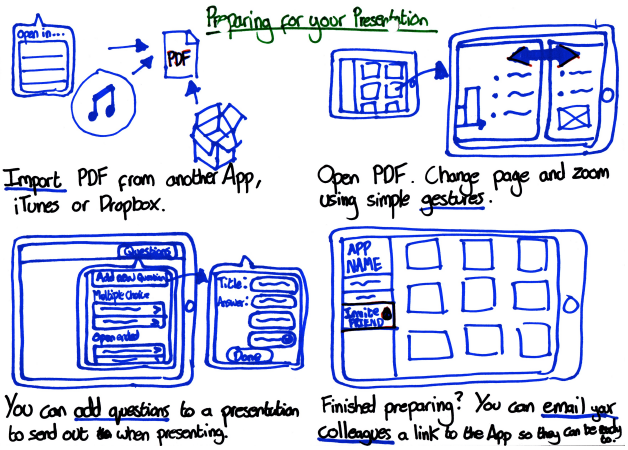


Figure 3.1: Preparing to present

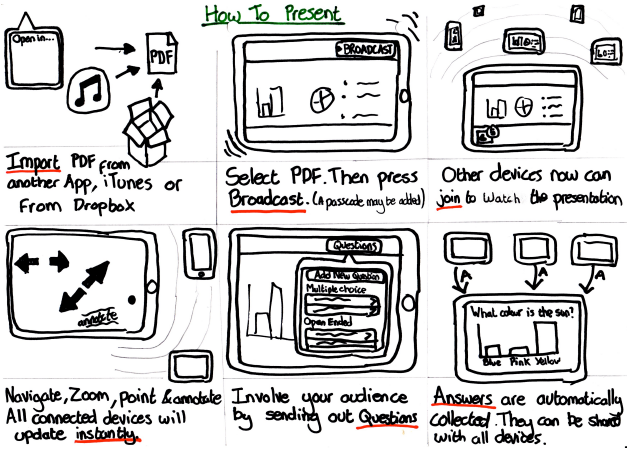


Figure 3.2: Presenting a document

questions to users (FR4, FR10). User's responses will automatically be collected and presented to the presenter. This storyboard uses many of the screens identified in the first storyboard and also requires the following screens:

- List of viewers
- Answers view

3.1.3 Viewing a presentation

Figure 3.3 shows the series of actions required to view a live presentation. The flow of the application is similar to that of a presenter except that the user has to select a presentation from the available presentations list. When the user selects the presentation it will automatically download to their device (FR2). Once downloaded, the presentation will open and display the same content as on the presenters device (FR1, FR3, FR5, FR6, FR17). During the presentation, a user can ask a question through the application to avoid interrupting the presenter (FR15). The presenter will receive an unobtrusive notification so they can answer the question at a convenient time. The presenter can also send out questions during the presentation (FR4, FR10). These question will instantly display on viewer's devices, allowing them to submit an answer. At the end of a presentation viewers are asked if they would like to save the presentation providing the presenter has given their consent (FR9).
This storyboard identified three additional screen:

- Available presentations list
- Ask presenter question screen
- Answer question screen

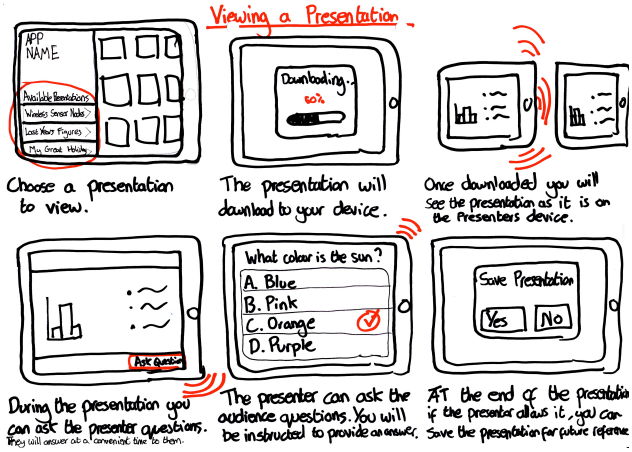


Figure 3.3: Viewing a presentation

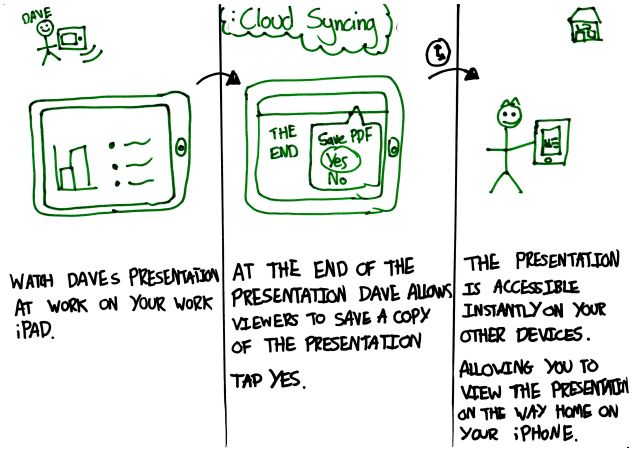
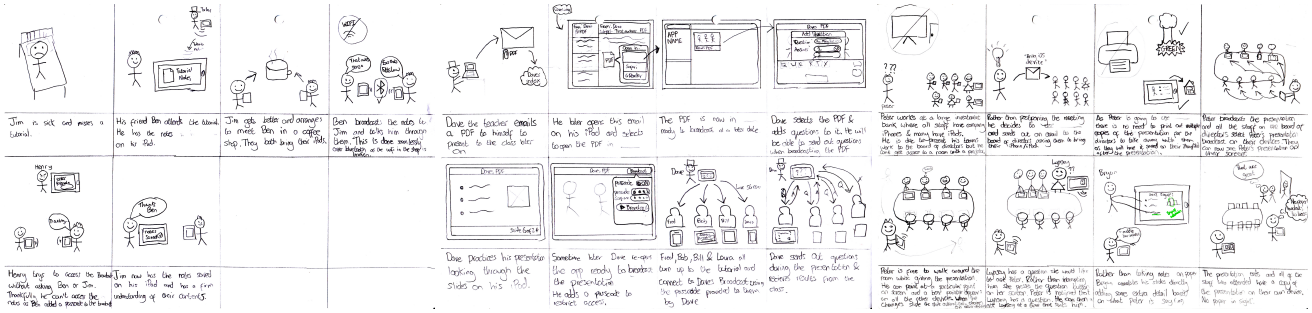


Figure 3.4: Document syncing

3.1.4 Document syncing

Figure 3.4 illustrates how document syncing will work between devices. When a new file is added to one device it is automatically synced with other devices, ensuring documents are always accessible to users (FR16). The storyboard demonstrates a user saving a document to an iPad at the end of a presentation (FR9) and viewing it later on an iPhone. No additional screens were uncovered through this storyboard.

3.1.5 Scenarios



(a) Personal use (b) Academic use (c) Business use

Figure 3.5: Detailed storyboards illustrating specific scenarios
(full size storyboards included in appendix D)

Personal scenario
Figure 3.5a gives an example of the application being used in a public area by two individuals: a presenter and a viewer. The presenter secures the broadcast with a passcode to limit who can access the presentation (FR14). This successfully stops an intruder from accessing the presentation without consent.

Academic scenario
Figure 3.5b shows how the application could be used within academia. Here, a teacher sends a document to himself and imports the file into the application using the 'open in' functionality (FR11). He then opens the document (FR1) and adds questions to it (FR4, FR10), which he will send out later when presenting. He reads through the document ahead of time to familiarise himself with its contents. Just before the class starts, he re-launches the application and broadcasts his document with a passcode, to ensure only his class have access to it (FR14). As the pupils arrive, they launch the application on their devices and join the live broadcast (FR1, FR2). He then talks the class through the document and sends out questions whenever appropriate, allowing him to determine if the class are having any difficulties with the lesson.

Business scenario
Figure 3.5c describes how the application could be used in a business environment, where it can often be difficult to book a room with a projector at short notice. The application is used to contact the attendees, advising them to install the application prior to the meeting. No paper handouts are printed as viewers are given the option to save the presentation (FR9). As the presentation contains confidential information, a passcode is added to the broadcast (FR14). During the presentation, a user raises a question via the application without disturbing the presenter (FR15). The question is stored on the presenter's device allowing it to be answered at a convenient time. At the end of the presentation, all viewers save the presentation so they can refer to it later.

The three scenarios described above, depend on the ability to add a passcode to a broadcast. This functionality will require a Broadcast Options Screen.

3.1.6 User Feedback

The storyboards were presented to several potential users from different backgrounds. Users were asked to walk through the storyboards and state if anything was unclear. All users were able to understand the storyboards and commented that they could relate to many of the situations. Users did not identify any screens that were omitted, and were able to follow the storyboard without asking questions, which implies that the flow of the application is logical. The passcode functionality proved popular with users in employment. They also noted the application would save them a lot of time at the printer. Students commented that they thought the application would make lectures more interactive.

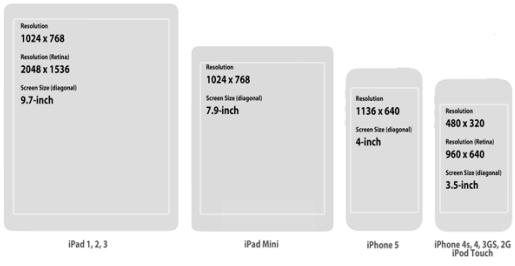
3.2 User Interface

This section describes the transition from requirements and key screens to detailed screen designs. Research into the platform was carried out to determine the platform conventions and constraints.

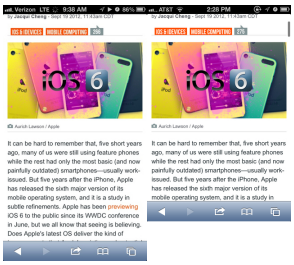
3.2.1 Interface Background

The main element of today's smartphone is a large touch screen display, which acts as a blank canvas, giving developers the freedom to innovate and create totally unique applications. Developers have published over 700,000 applications[11], many of which perform similar tasks. In this competitive market, where applications can be installed within minutes (and removed within seconds), users have come to expect elegant design combined with efficient usability. If an application does not meet this criteria, there will be little incentive to keep the application, as there is likely to be a similar application readily available in the App Store.

In order to create an application which will be valued by users, attention to detail is crucial. To help developers, Apple have published a set of Human Interface Guidelines[1], which are based on the way people think and interact with mobile devices. These guidelines discuss common standard human interface principles (such as consistency and feedback) as well as platform specific principles. One extremely important principle is getting to grips with the platform's unique characteristics: screen size, orientation and gesture support. Figure 3.6a shows the four different screen sizes which iOS currently runs on.



(a) iOS screen comparison



(b) Safari running on iPhone 5 and iPhone 4

Figure 3.6: iOS screen differences

Although there are four different screen sizes, they can be split into two main categories: tablet (iPad and iPad Mini) and phone (iPhone and iPod Touch). The iPad and iPad mini both share the same screen resolution and aspect ratio, which means applications look identical on each device. The iPhone 5 has larger screen compared to all previous iPhones (and iPod touches). However, as the width of the screen remains the same, developers can create just one interface for all iPhones and apply minor alterations to optimise it for the taller screen. Figure 3.6b illustrates how Safari (a web browser) makes use of the extra screen real-estate to show the user content, without modifying the design.

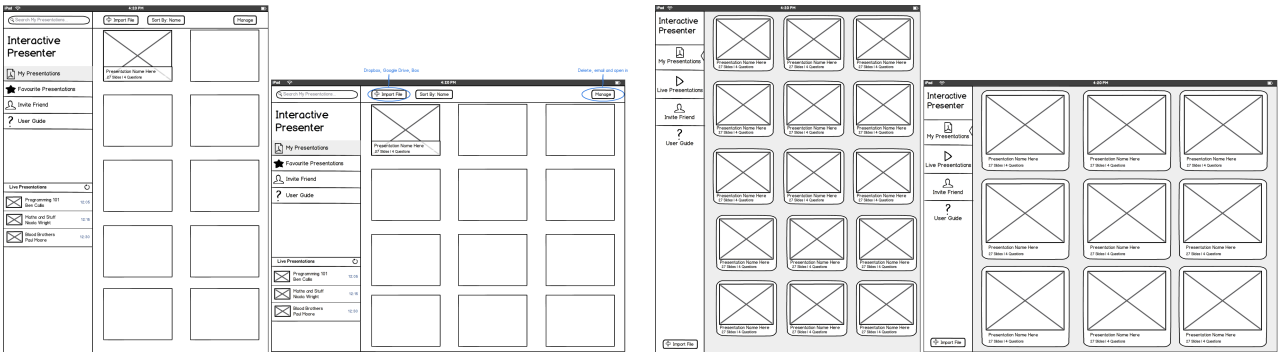
[1] Apple Human Interface Guidelines - `http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG`

Several iOS devices now include a high density “Retina display”, which has four times more pixels than their non-retina counterparts. Although the pixel count is dramatically higher on retina displays, there is no need to create a custom interface for these devices, as the screen size and aspect ratio is identical to the low density versions. However, applications should include multiple graphical assets optimised for each display. For example, a 44 x 44 pixel image designed for a non retina display, will need to be 88 x 88 pixels for the retina display, to avoid visible pixelation whilst maintaining the same apparent dimensions.

3.2.2 iPad Wireframes

Following on from identifying the flow of the application and considering the platform constraints, wireframes were created for the major screens. Wireframing is a interface planning process which takes place before actual application development. It involves planning the layout of the app, making it easier to visualise where features will be accessible. Wireframes also help when it comes to the development phase, as they are broken down into separate views (screens), which helps separate the development into distinct steps. Different levels of detail can be used when wireframing, which can be broken up into two categories in terms of fidelity, or how closely they resemble the finished product. In order to gain feedback from potential users a high-fidelity approach was taken, with a high level of detail that closely follows the actual design of the application. Despite using this approach, the wireframes were purposely made to look like rough sketches, to encourage users to give honest feedback.

During the design stage button dimensions were carefully considered to ensure that they are easy to touch with a human finger. The Human Interface Designs guidelines recommend keeping touch targets above 44x44 pixels. This particular guideline has not been revised despite the launch of the iPad mini, which has the same amount of pixels as the original iPad into a smaller display. To ensure targets are easily tapped on all iPads, a minimum of 50x50 pixels has been used in all designs.



(a) Design 1 (b) Design 2

Figure 3.7: iPad Main Screen Wireframes

Main screen - Design One

The main goal for this design was accessibility, focusing on reducing the number of steps required to perform tasks. Figure 3.7a shows the main screen presented to the user when they launch the application. The screen is divided into two panels: a navigation panel (*left*), providing access to other screens and live presentations; and a large detailed panel (*right*), which shows content relating to the selected item in the navigation panel. When the application is launched ’My Presentations’ is selected by default, allowing the user to instantly access all the presentations stored on their device. Each presentation is represented by a large thumbnail (taken from the first slide) and some accompanying descriptive text. The descriptive text includes the presentation name, number of slides and the number of questions attached to the presentation. The design also included a toolbar along the top edge of the screen, which allows users to search, sort and manage their documents.

When the user holds the device in portrait orientation, the panels are automatically resized. The left panel maintains its width to ensure that live presentations are still easy to locate. The right detail panel adjusts to display two columns of cells, allowing the cells to maintain their dimensions.

Main screen - Design Two

In the second design menus and other controls are kept to a minimum, allowing users to focus on their documents (see figure 3.7b). Each document is represented by a large cell, containing a thumbnail and a textual description. A single menu is located along the left hand edge of the display, giving users quick access to different views. A list of live presentations can be viewed by selecting the second tab from this menu. Live presentations are also displayed visually in grid format, keeping the interface consistent. The menu maintains its position along the left hand edge of the display in both orientations, to ensure the menu remains within the zone reachable with an average thumb-span (illustrated in figure 3.8c and 3.8c). As the tabs within the menu are located within this zone, users are able to comfortably switch tab with their left thumb whilst holding the device (shown in figure 3.8b). The menu provides similar functionality to the default tabbar view which is part of the iOS SDK. However, the default tabbar can only be displayed along the bottom of the screen (see the bottom of figure 3.8c) which can be difficult to reach on the iPad.

(a) Two hands along bezel (b) Left hand along bezel (c) Left thumb reach

Figure 3.8: Custom tab bar designed based on thumb reach when holding iPad

In both orientations the document grid contains three columns. This allows users to locate their documents based on their position in the list. To achieve this, cell dimensions are slightly smaller in portrait orientation.

Document cells

Figure 3.9 illustrates three cell styles, which could be used in either of the main screen designs.

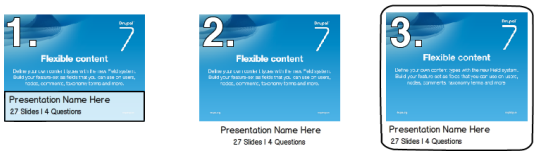


Figure 3.9: Three different cell styles used to represent a document.

Document Viewer screen

The document viewer screen, shown in figure 3.10, allows users to view and navigate documents. Presenters and viewers have access to different functionality from the top toolbar (shown in 3.10). For example, the presenter has access to a list of questions which can be sent out to all viewers. This button is not in the menu for viewers as the functionality is presenter specific.

To make presentations more immersive, all controls can be hidden/shown by tapping the screen (see figure 3.10c). A page scroller is situated within the bottom toolbar, allowing users to quickly change page. Users can also navigate a document through direct manipulation, by using a swipe gesture (illustrated in figure 3.10d).

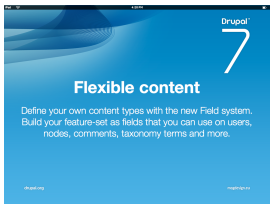
The screen also allows users to configure and start broadcasting their document, by selecting the 'Broadcast' button (located in the upper right corner of the presenter view). Once a document has been broadcast other users can join, allowing them to watch the presentation. During a broadcast, presenters can access a list of current viewers and their questions, by tapping the 'broadcast' button (shown in figure 3.10e).

Question screens

From the document screen, users can access the questions associated with the presentation, by selecting the 'Questions' button (situated in the top toolbar). This screen presents existing question in a list and allows users to add additional questions. Users can select existing questions from within the list to reveal their full details and

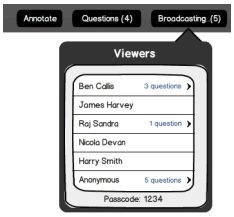
(a) Presenter View

(b) Viewer view



(c) Full screen

(d) Page changing



(e) List of viewers visible to presenter

Figure 3.10: Document Viewer

send them out to viewers by selecting the 'Send' button. When a question is distributed to viewers, an additional screen is displayed, allowing the presenter to monitor the audience's responses. This screen includes a graph to give a visual representation of the results. When a viewer receives a question, it is automatically presented with the possible answers, allowing them to submit an answer to the presenter.

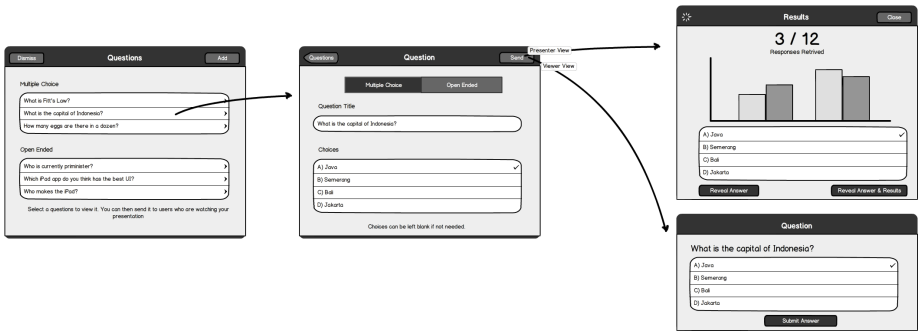


Figure 3.11: Question screens for presenter and viewers.

3.2.3 User Feedback

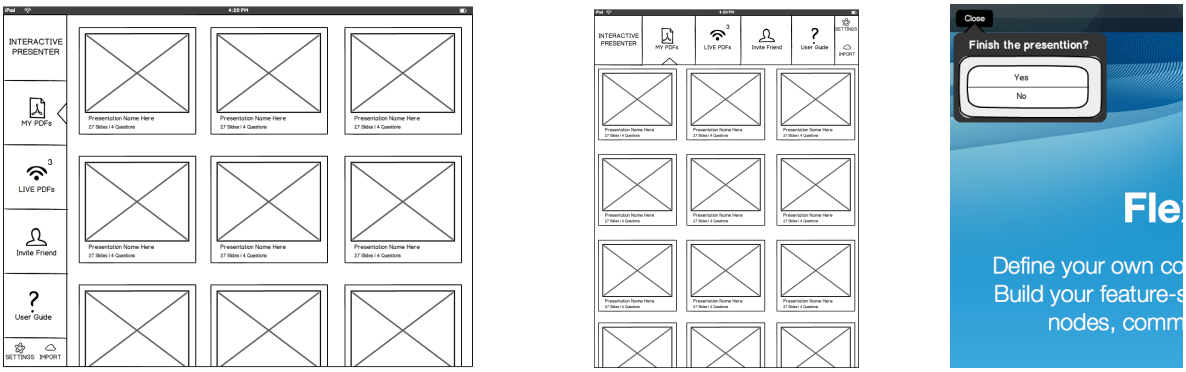
The designs were presented to six potential users to gain feedback, allowing a final design to be constructed.

Design one
Users liked how it was possible to instantly see how many live presentations were currently available. However, many stated that this section was too small. Users also commented on the amount of wasted space along the left panel in portrait orientation. They noted that only 10 documents were visible in portrait orientation, compared to 12 in landscape. Based on this, they stated that they would be less likely to use the application in portrait mode.
Design two
Users preferred the smaller menu bar, as it allowed them to focus on their documents without being distracted by large menus and controls. Users liked how live presentations were displayed in a similar way to their local documents. It was noted that live presentations could easily go unnoticed, as they are not instantly visible when the application is first launched. As with design 1, users commented on the amount of wasted space along the menu bar when the application is in portrait mode.

Cell Styles
Overall, cell style 3 was most preferred, although negative points were raised about all three styles. One user commented that placing text over the thumbnail in style 1 could obscure distinctive parts of the thumbnail, making it difficult to identify documents. Users also asked if tapping the text in style 2 would perform the same actions as tapping the thumbnail. The curved border used in style 3 was criticised as it did not match the straight lines used by the thumbnail.
Document Viewer
Feedback received for the document viewer screen was extremely positive. Users particularly liked how it was possible to hide the toolbars with a single tap. One user noted that the close button should prompt the user for confirmation, to ensure that a presenter doesn't end the presentation by mistake. Users also asked if it would be possible to remove the status bar to give more room to the document.
Question Screens
Feedback was towards the question screen was extremely positive. Users appreciated how simple it was to add and distribute questions.

3.2.4 iPad Final Design

A final design was developed, taking into account the user feedback and using concepts from the original designs. Figure 3.12 shows the final main screen design, which combines the accessibility of design one with the simplicity of design two.



(a) Landscape (b) Portrait (c) Close prompt.

Figure 3.12: Final Design - Menu sticks to smallest edge in both orientations

The menu bar has seen the biggest change in the final design, addressing the wasted space issue (reported by users). The height of all buttons on the menu have been increased to make use of all the available space. When the device is rotated into a portrait orientation, the menu sticks to the short edge of the device and all buttons rotate (see 3.13b), using all available space. Moving the menu to the top in portrait orientation leaves more space for the document grid, allowing three columns to be displayed in both orientations with just a small change to the cell size. Design two placed all menu buttons within reach of the users left thumb. Increasing the size of the buttons makes it more difficult to reach the buttons located right at the bottom of the menu. However, the two most frequently used buttons: 'My PDFs' and 'Live PDFs' are positioned within the thumb reach zone, maintaining quick access. Based on the feedback from design one, a badge is included in the 'Live PDFs' menu button to indicate the number of live broadcasts currently available.
The document viewer was also slightly modified based on user feedback. Figure 3.12c shows the final document viewer, with the addition of a confirmation popover triggered by tapping 'close' button. The status bar has also been removed, allowing documents to fill the entire screen.

After finalising the wireframes, work began on the visual aspects of the application. To experiment with different colours and textures, a vectorised version of main screen was designed in Adobe Fireworks [2]. This

[2]Adobe Fireworks - `http://www.adobe.com/products/fireworks.html`

made it possible to experiment with different design by applying colours and textures to individual vectors.

(a) Final design including graphics

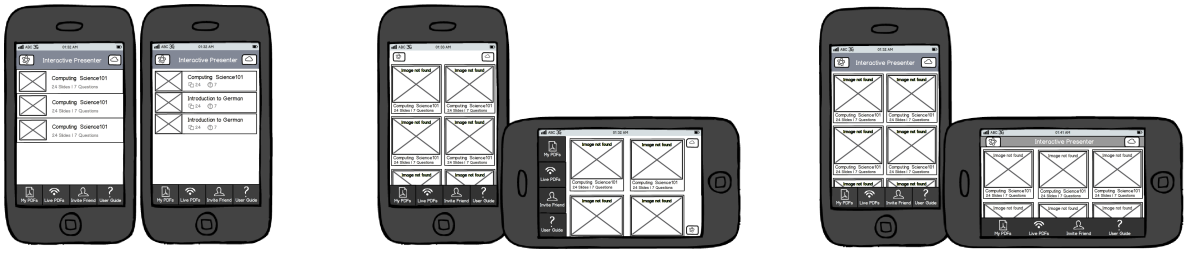
(b) Vectorised design created in Fireworks

Figure 3.13: Final detailed design

Figure 3.13a shows the complete design which uses five bold colours to give the application a fresh identity. Each button in the menu bar uses a different colour background which outlines the size of the button and allows users to identify a screen with a certain colour. Each cell has a white background with a slight drop shadow to add a sense of depth to the documents. Cell titles are styled with the same colour as the active tab, to link the documents grid with the bold colour palette.

3.2.5 iPhone Wireframes and Detailed Design

After finalising the iPad user interface design, work began on design the user interface for the iPhone. The design needed to provide access to all functionality of the iPad version on a significantly smaller screen. To achieve this many iPhone applications only support portrait orientation, due to the limited height available in landscape. This approach is not viable for this application as many documents will be best suited for landscape viewing. Having just some screens support landscape orientation would damage the flow of the application, requiring the user to change the way they hold their device frequently. As a result, it was decided that the main home screen should also support landscape orientation.



(a) Designs using a table view

(b) Design with iPad style menu

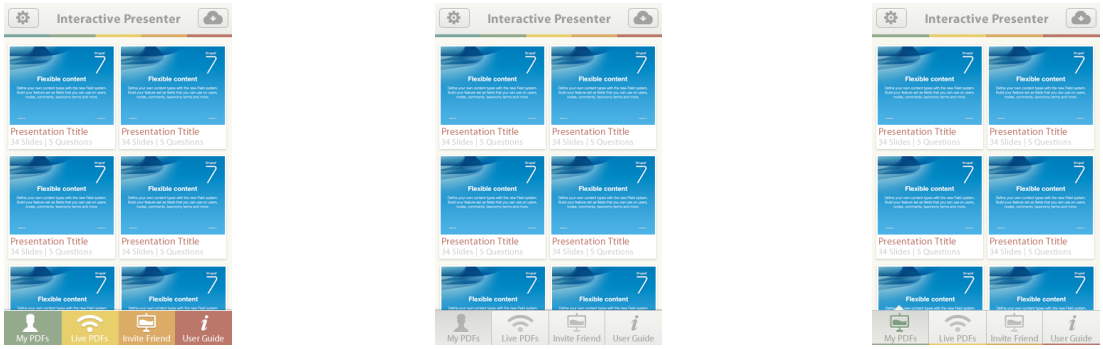
(c) Final design standard tabbar

Figure 3.14: Different iPhone designs.

A naive design approach would be to simply scale the iPad version to fit the smaller iPhone screen. However, this would create a frustrating user experience, as many onscreen targets would be smaller than the recommended 44x44 pixels. Figure 3.14 shows three different designs, each making use of slightly different UI components. Due to the limited width available, one design was created using a table to present the documents (shown in figure 3.14a). In this design, rows span across the whole width of the device acting as large touch targets. Due to the limited height available in each cell, thumbnails have to be considerably smaller, making the textual labels more significant. Although this design is practical and extremely common in many iPhone applications, it is not consistent with the iPad version.

To improve consistency across devices, a second design was created which used the grid view found in the iPad version (shown in figure 3.14b). As with the iPad version the menu bar is attached to the shorter edge of the device in both orientations. However, there are two noticeable differences. Firstly, the settings and import buttons are positioned at the opposite side of the screen due to the limited amount of width available on the device. Secondly, the menu bar is located at the bottom of the screen when the device is in portrait orientation, to make it easier to reach. This menu bar looked extremely similar to the standard iOS tabbar when used in portrait

orientation. However, as previously mentioned the standard tabbar is always located at the bottom of the screen in both orientations. As the menu bar appeared similar to the standard tabbar it was felt that repositioning it in landscape orientation could confuse users.
Figure 3.14c shows the final iPhone design, which builds on the two previous designs. A standard iOS tabbar is used in this design, allowing users to be instantly familiar with the application. The import and settings button are located in a navigation bar at the top of the screen, which also displays the application name. In portrait orientation the tabbar stays at the bottom of the screen allowing users to quickly change tabs whilst holding the device. The increase width in portrait orientation allows the number of columns in the grid view to increase to three, making the most out of the limited screen real-estate.



(a) iPad styled tabbar (b) Plain tabbar (c) Final design

Figure 3.15: iPhone detailed designs

The same palette and custom graphics were used when creating the detailed design. Figure 3.15a shows the initial design which includes a styled tabbar mimicking the menu bar found on the iPad version. Whilst the use of bold colours worked well on the iPad, it was decided that excessive use of these colours on a smaller display put to much focus on the tabbar. Based on this, a second design was created with a single colour tabbar (see figure 3.15b). Although limiting the use of colours in this design helped emphasise the importance of the documents contained in the grid above, consistency between the two versions was reduced. To improve consistency across devices, a final design was created making additional use of the bold colours (illustrated in figure 3.15c). This design builds on the second design, by adding an additional thin colour strip to the bottom of the tabbar, giving each tabbar item a unique colour corresponding to the iPad version. The selected tab is represented by the addition of colour to the icon as well as a subtle change to the background. A small triangle pointer has also been added to indicate which tab is selected, similar to the one found in the iPad design.
This design also works with the iPhone 5, as the menus are anchored at the top and bottom of the screen, allowing the grid of documents to fill the rest of the display.

3.3 System Structure

The user interface is the central component of the overall system. In order for the application to satisfy all of the requirements outlined in section 2.3, four additional components are required: Document Viewer, Network, Question Manager and Data Store. Each components is described below.

3.3.1 Document Viewer

To enable users to present and view presentations, a Document Viewer is required. The Document Viewer must render individual pages and allow users to navigate though the document. This component will interact with three other components in the system: rendered pages will be sent to the UI component to be displayed to the user, navigation events will be sent to viewers via the networking component and documents will be read from the backend storage. Navigation events will be triggered when a presenter changes page, zooms in on the document or uses the laser pointer. The events will be forwarded to all viewers to keep all displays synchronised. Events

will be triggered by the following gestures:

Left swipe or right edge tap	Move to next page	**Pinch-out Pinch-in**	Zoom in/out
Right swipe or left edge tap	Move to previous page	**Tap and hold**	Laser pointer

3.3.2 Network

In order for devices to communicate together, a network needs to be formed between them. The two most common networking architectures: client-server and peer-to-peer are shown in figure 3.16.

(a) Client-Server topology (b) Peer-to-peer

Figure 3.16: Common network architectures

A client-server network involves multiple clients connecting to a single server. The server is in charge of all shared data and can communicate with all clients. However, clients can only interact with the server. Peer-to-peer networks impose no such constraints. There is no dedicated server and all clients can communicate with each other. The client-server architecture is a perfect fit for the application, as only one device in a connected network can take the role as a presenter. When the application is launched, the device will act as a client and automatically search for servers. A device will become a server when they present a document. The presenter will then be able to transmit the document to clients upon request and send navigation events to viewers to keep the presentation synchronised. Once a client has joined a presentation it will be able to communicate with the server, allowing questions to be raised.

3.3.3 Question Manager

To allow presenters to send questions to viewers a question managing component is required. This component will allow users to add, edit and view questions attached to a document. The component should allow users to add questions before and during a presentation. To allow questions to be persistently stored the component will need to interact with the Data Store. During a presentation, the component should communicate with the networking component to send questions to viewers. The component will also interact with the user interface to present viewer answers in a chart, giving the presenter a visual overview of the results.

3.3.4 Data Store

For documents and questions to be available between application launches, they need to be stored in persistent memory on the device. This can be achieved with a database and a folder with write permissions (on the device). Each document will correspond to one entity in the database. Each entity will have several attributes: title, thumbnail path, file path, number of pages and a set of questions. The actual file and thumbnail will be stored on the devices file system rather than directly in the database, to avoid converting between data types. Questions and relating answers will also be stored in the database. An entity relationship diagram was created to illustrate the entities required in the database (figure 3.17). The diagram illustrates the cardinality of the relationship between entities. A document can have many questions, which can have many possible answers.

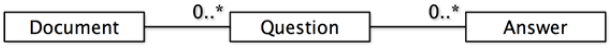


Figure 3.17: Entities required in the application

Chapter 4

Implementation

This chapter describes two implementation cycles and a user evaluation which took place between the cycles. The initial phase included implementing all 'must have' and the majority of 'should have' requirements. A user evaluation was performed after the initial development cycle to gain feedback and further refine the functional requirements. The final implementation phase was based on the user feedback and fulfilled all the remaining 'should have' and a large majority of the 'could have' requirements.

4.1 Initial Implementation

4.1.1 Home Screen

Implementation began with the creation of the home screen. In order to allow the application to adapt to the current device, different code and views need to be loaded at runtime. This is achieved by wrapping device specific code in conditional statements, using the `[UIDevice currentDevice]` method. As this check is used frequently throughout the application, macros were created to indicate the current device type.

```
1 #define deviceIsIpad ()          ([[ UIDevice currentDevice ] userInterfaceIdiom ] == UIUserInterfaceIdiomPad)
2 #define deviceIsIPhone ()        ([[ UIDevice currentDevice ] userInterfaceIdiom ] == UIUserInterfaceIdiomPhone)
```

These macros were added to a constants files (`TPConstants`), which is imported into the applications prefix file[1] so it is automatically imported by into all classes throughout the project. The iOS platform loads device specific resources at runtime based on naming conventions. The '`@2x`' suffix can be used to denote resources for retina display devices. Device specific recourses can also be defined using the '`~ipad`' and '`~iphone`' suffix. For example, an interface file (XIB[2]) named `IPViewController~ipad` would automatically load on an iPad when a request for `IPResultsViewController` is made.

When the application is launched the `applicationdidFinishLaunchingWithOptions:` method is invoked within the application delegate to create and present the first screen to the user. The code below illustrates how this method uses the macros to configure the view hierarchies accordingly.

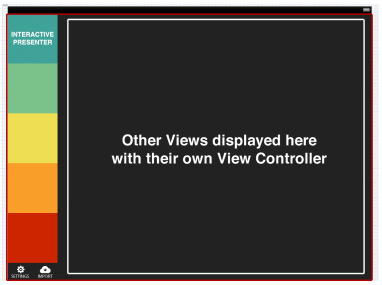
```
1 - (BOOL)application:(UIApplication *) application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
2     self.window = [[UIWindow alloc] initWithFrame:[[ UIScreen mainScreen] bounds]];
3     if (deviceIsIPhone () ) // macro defined in constants file
4         [ self setupiPhoneViewHiearchy]; // configure iPhone view hierarchy (uses tabbar )
5      else // must be iPad
6         [ self setupiPadViewHiearchy]; // configure iPad view hierarchy using custom navigation style
```

[1] prefix file - a project specific precompiled header

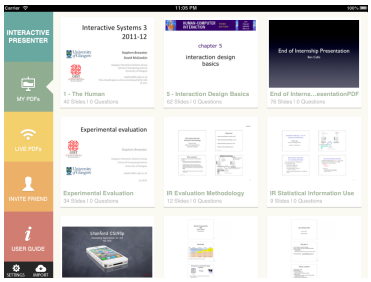
[2] XIB files (also known as NIB files) are used to create interfaces graphically using Xcode - `https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/LoadingResources/CocoaNibs/CocoaNibs.html`

iPad

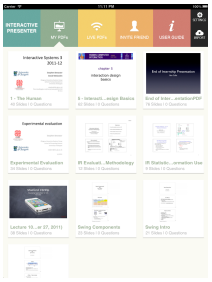
The iPad view hierarchy revolves around the custom navigation bar. To implement this, a view controller[3] (`IPCustomNavigationViewController`) was created to control the menu buttons and manage other view controllers. A visual representation of this view controller is shown in figure 4.1a). Each menu button in the navigation bar is a custom view containing a title, icon and badge (illustrated in figure 4.2a). When a button is pressed, the corresponding view controller is loaded and its view is presented in the detail view. To preserve memory on the device the current view is unloaded before the new one is presented.



(a) Custom navigation VC



(b) Landscape orientation



(c) Portrait orientation

Figure 4.1: iPad main screen with custom navigation bar fixed to the smallest edge

To implement the rotating menu bar, the `IPCustomNavigationViewController` overrides the `willAnimateRotationToInterfaceOrientation:` method which is invoked on rotation of the device. In this method the frames[4] of all onscreen views are adjusted to their correct position. The frames for each view are stored in the constants file, making them easy to modify in the future if additional devices are released.

```
1  - (void) willAnimateRotationToInterfaceOrientation :( UIInterfaceOrientation ) toOrientation duration :(NSTimeInterval)d{
2      // tell all buttons to adjust
3      [_btnMyPDFs adjustForOrientation: toOrientation ];
4      ...
5      // sort out the main view frame and other buttons
6      if ( UIInterfaceOrientationIsPortrait ( toOrientation )) {// Portrait frames
7          _detailView .frame = kGridViewFramePort; // frames defined in constant file
8          _btnPreferences .frame= kButtonSettingsFramePort;
9          _btnImport .frame=kButtonImportFramePort;
10     }else { .. }        // Landscape frames
11 }
```

To ensure the selected menu button indicator (triangle point) remains attached to the details view additional layout logic was required to transform the view.

```
1  -(void) adjustForOrientation :( UIInterfaceOrientation ) interfaceOrientation {
2      if ( UIInterfaceOrientationIsPortrait ( interfaceOrientation )) {// Portrait
3          self .frame = CGRectMake(_btnNumber * BUTTON_WIDTH , 0, BUTTON_WIDTH, BUTTON_HEIGHT); // move button frame
4          _selectedTriangle .transform = CGAffineTransformMakeRotation(90 * M_PI/180); // rotate trainlge point so it faces the detailed view
5          _selectedTriangle .frame = kButtonSelectedTriagnleFramePort; // move triangle point
6      } else {...} // Landscape
7  }
```

Figure 4.1b illustrates the 'MY PDFs' menu button selected in the custom navigation bar. When this button is selected the `IPMyFilesViewController` is loaded and its view is displayed in the detailed view. The `IPMyFilesViewController` displays the user's documents in a grid layout using GM-Grid-View[5]. GM-Grid-View requests content from its delegate (`IPMyFilesViewController`), which is responsible for creating the individual cells and responding to user interactions, such as cell selections. Figure 4.2b shows the cells custom view, which is created and configured by the delegate. In order to minimise memory use, the Grid view reuses any existing cells in memory before instantiating new cells.

[3] View controller objects manage communication between view objects and model objects.

[4] A view's frame is the position of its rectangle in the superview's coordinate system. It is made up of four floats: x position, y position, width and height.

[5] An open source performant Grid-View for iOS (iPhone/iPad) - https://github.com/gmoledina/GMGridViewreadme

(a) Menu Button View

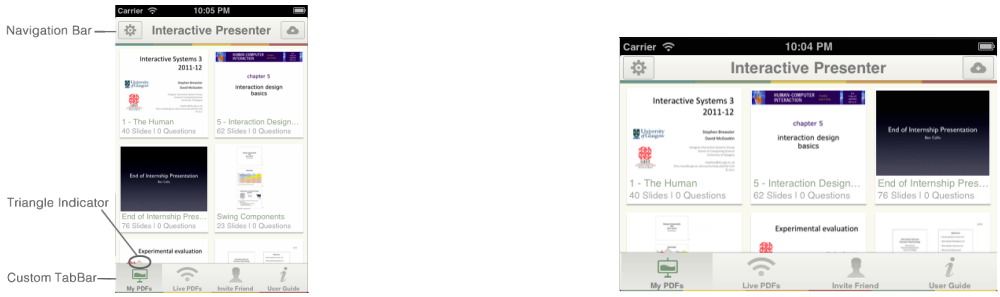
(b) Grid Cell View

Figure 4.2: Custom Views

Figure 4.2b illustrates the custom cell view, which contains a document thumbnail, title and description label. Each thumbnail is generated from the first page of the document and stored in persistent storage for future access. This is done in a background thread to keep the UI smooth and responsive.
When the orientation of the device changes, the cell dimensions are modified to allow three columns in both orientations.

iPhone

The iPhone navigation hierarchy revolves around a `UITabBarController`, which is part of the Cocoa Touch framework[6]. This controller manages multiple other view controllers, which users can access by selecting a tabbar item. Rotation is handled automatically and the tabbar stays situated along the bottom of the screen in both orientations (illustrated in figure 4.3).



(a) iPhone Landscape

(b) iPhone Portrait

Figure 4.3: iPhone main screen with the tabar managing multiple ViewControllers.

Each tabbar item contains an icon, label and an optional notification badge. The tabbar is customised using the `UIAppearance` protocol[7], which allows global appearance properties to be specified. The `UIAppearance` protocol creates an appearance proxy, which is used to modify the appearance of all subsequent instances of that class.

```
1 [[UITabBar appearance] setBackgroundImage:[[UIImage imageNamed:@"tabbarBg"] resizableImageWithCapInsets:UIEdgeInsetsZero]];
2 [[UITabBar appearance] setSelectionIndicatorImage :[[ UIImage imageNamed:@"tabbarSelectedTabBg"] resizableImageWithCapInsets:UIEdgeInsetsZero]];
```

To implement the triangle indicator (shown in figure 4.3a), a subclass of `UITabViewController` was created, which overrode several methods, including the `didSelectViewController` method which is called when a new tab is selected. This method adjusts the triangle indicator's x co-ordinate, positioning it above the selected tabbar item. To provide a visual fluid transition between states, the indicator's frame is adjusted within an animation block[8]. The code for this method is shown below.

```
1 - (void) tabBarController :( UITabBarController *)theTabBarController didSelectViewController :( UIViewController *)vc{
2     CGRect frame = tabBarArrow.frame;
3     frame. origin .x = [ self horizontalLocationFor : self . selectedIndex ]; // gets the centre x coordinate of the selected tab
4     [UIView animateWithDuration:0.2 animations :^{ tabBarArrow.frame = frame; }];
5 }
```

[6]Coca Touch is a UI framework created by Apple

[7]A protocol is a list of method declarations that are not bound to any particular class.

[8]Block objects are a C-level syntactic and runtime feature.

To include a navigation bar at the top of the screen, each view controller within the tabbar is embedded within a `UINavigationController`. This allows each view controller to set a title and add buttons to the navigation bar. The navigation bar's background and text colour is styled using the `UIAppearance` protocol.

The bar buttons located within the navigation bar, are customised with a background image (figure 4.4a). As the image contains a gradient and border, it is not possible to alter the image's width without causing the button to appear stretched (figure 4.4b). To allow the background image to be used for buttons of any size, cap insets are applied to the image using the `resizableImageWithCapInsets:` method. The insets are applied using the UIEdgeInsets structure, which specifies the top, left, bottom and right insets to apply to an image. During scaling or resizing of the image, areas covered by a cap are not scaled or resized (figure 4.4c.

(a) Scaleable button image

(b) Stretched button

(c) Scaled button

Figure 4.4: Custom resizable navigation button

```
1 #define kNavBarButtonImageCapsets                UIEdgeInsetsMake (3, 3, 3, 3)
```

```
1 [[UIBarButtonItem appearance] setBackgroundImage:[[UIImage imageNamed:@"navButton"] resizableImageWithCapInsets:kNavBarButtonImageCapsets] forState:
        UIControlStateNormal barMetrics:UIBarMetricsDefault];
```

As the background images border is 3 pixels on each edge, all insets are set to 3, to ensure that these pixels are not resized when scaling the image. A button using these insets is shown in figure 4.4c. The complete iPhone home screen is shown in figure 4.3.

4.1.2 Document Viewer

As identified in section 3.3 the application needs a component to display documents. Within the Cocoa Touch framework it is possible to render several common document types[9], including PDF, Doc and PTT, using a `UIWebView`. However, as the view was created for accessing web content, navigation controls are limited. This is apparent when displaying PDF and PPT documents, as all pages are displayed vertically. Although, this may be adequate for displaying textual documents, it is restrictive for displaying full page presentations. An example of a `UIWebView` displaying a full screen PDF presentation is shown in figure 4.5.

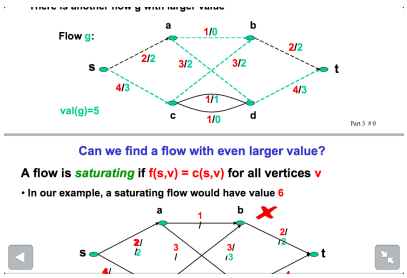


Figure 4.5: UIWebView navigation

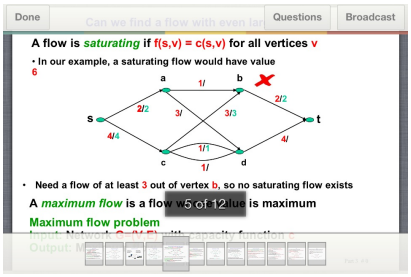


Figure 4.6: Customised VFR Reader

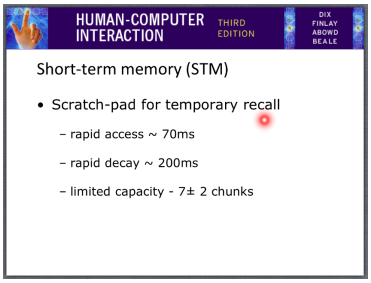


Figure 4.7: Laser pointer

As the `UIWebView` would not allow for all the gestures identified in section 3.3.1, several alternatives were considered, before deciding on VFR Reader[10]. VFR Reader is a universal open source PDF library which supports iOS 4+. Although the library only supports PDF documents, it was considered the best choice due its full screen rendering capabilities and extensive navigation features. PDF is also a well known commonly used format, which many office applications export to. The PDF library allow users to change page in a document with horizontal swipe gestures or via a page scroller situated along the bottom of the display (illustrated in figure 4.6).

[9] UIWebView supported file types - `https://developer.apple.com/library/ios/#qa/qa2008/qa1630.html`
[10] VFR Reader - `https://github.com/vfr/Reader`

Modifications
Although the library was able to render PDFs without any changes, several modifications were made to customise its appearance and increase its extensibility. As the library supports iOS 4 it uses manual memory management, rather than ARC[11]. To make the library easier to extend and maintain, all memory management was converted to automatic reference counting

When displaying a PDF, the library overlaid all controls onto toolbars at the top and bottom of the screen. These toolbars were not the standard toolbars from the Cocoa Touch framework. This made it difficult to customise as it could not interact well with other UIKit components and UIAppearance customisation was not automatically applied. Based on this, the library was refactored, replacing all instances of the custom toolbar with the standard `UIToolbar`. Buttons were added to the top toolbar to support broadcasting and questions which are discussed in later sections. The final document viewer running on an iPhone is shown in figure 4.6.

Gestures
The PDF library includes page changing gestures, which animate the transition between pages. This animation was removed when using the left or right side edge of the screen to change page, to allow users to notice subtle changes between slides in a presentation. To do this the increment and decrement page methods were modified to accept an additional boolean argument, which is used to determine if the page change should be animated. When swipe gestures are used to change the page, the method is invoked with animated set to `YES`. When the screen edges are tapped the method is called with animated set to `NO`.

```
1 - (void)incrementPageNumberAnimated:(BOOL)animated{ // called when changing page with a swipe gesture or by tapping the edge of doc
2       ...
3       [theScrollView setContentOffset:contentOffset animated:animated]; // change page with the animation if animated variable is YES
```

The library also includes a single tap gesture to toggle the visibility of the navigation controls and a pinch gesture to zoom in on documents. However, to implement all gestures defined in section 3.3.1, an additional gesture recogniser was required for the laser pointer. iOS allows gesture recognisers to be added to any view and allows several options to be set. The laser pointer was implemented by adding a long tap gesture to the document viewer. A dependency relationship is defined between the long tap gesture and the single tap gesture, to ensure they are both not trigged for one action.

```
1 UILongPressGestureRecognizer *longPress = [[UILongPressGestureRecognizer alloc] initWithTarget:self action:@selector(handleLongTap:)];
2 longPress.delegate = self;
3 [longPress requireGestureRecognizerToFail:singleTapOne]; // don't want single tap to occur as well
4 [self.view addGestureRecognizer:longPress];
```

When this gesture is recognised, the `handleLongTap` method is invoked. This method checks the state of the gesture to determine if the laser pointer should be shown or removed.

```
1       switch (tapGesture.state)
2           case UIGestureRecognizerStateChanged:
3               [self showLaserPointerAtAbsolutePoint:[tapGesture locationInView:self.view]]; break;
4           case UIGestureRecognizerStateEnded:
5               [self hideLaserPointer]; break;
```

The laser pointer is presented by adding an image to the document viewer, with an alpha transparency of 0.8 to make it appear more realistic. To ensure the point is not obscured by the user's finger, the image is positioned slightly above the touch point. This method is listed in appendix G.1 on page 97. When the gesture ends, the laser pointer fades out over a small time period. This is achieved by setting the alpha transparency value to zero in an animation block. The laser pointer is shown in figure 4.7.

4.1.3 Data Model

As discussed previously in section 3.3.4, the application needs to store documents, questions and answers to ensure they are available between launches. iOS has a comprehensive collection of tools and frameworks for

[11] Automatic Reference Counting combines the simplicity of garbage collection with the efficiency of manual memory managemen

storing and accessing data. It includes the SQLite[12] library, a low level relational database engine which is used in many platforms. Built on top of this is the Core Data framework which provides object-relation mapping, to integrate well with object-orientated programming. This framework provides an additional level of abstraction, removing the need to write raw SQL queries. Core Data objects can be created visually within Interface Builder and data can be fetched, updated or added using Cocoa Touch APIs. Core Data supports a range of common types, including string, date, and number. It is also possible to store custom types in Core Data by serialising them to raw binary data. However, this adds an extra overhead as data needs to be converted between types when reading and writing. To avoid this, PDFs and thumbnails are stored on the file system, within the application's sandbox[13], with their paths stored in the Core Data model. Each iOS application has its own sandbox, which contains three folders: *Documents*, *Library* and *tmp* (illustrated in figure 4.8). These directories constitute the applications primary view of the file system.

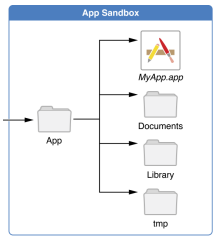


Figure 4.8: iOS app sandbox

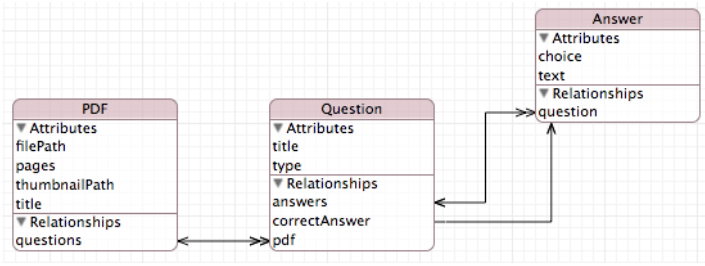


Figure 4.9: Database model.

Contents within the *Documents* directory are automatically backed up to iTunes when the device is connected to a computer, allowing its contents to be restored if the device fails. The *Library* directory is not backed up; however, data stored in this folder persists between runs of the application. Files kept within the *tmp* directory are only stored temporarily and the operating system may purge them when the application is not running. To ensure PDFs persist between application launches and are recoverable in the event of a device failure, they are stored within the *Documents* directory. PDFs received as part of a presentation are stored in the *Library* directory during the presentation and are deleted or moved into the *Documents* directory at the end of the presentation (discussed in section 4.1.4).
As thumbnails can easily be regenerated, they are stored within the *Library* directory, to ensure they are not backed up to iTunes. This directory is never purged by the operating system, so the thumbnail paths stored in the data model will always be valid.

The schema for the Core Data model is shown in figure 4.9. The model contains three separate entities: `PDF`, `Question` and `Answer`. The `PDF` entity consists of four attributes which contain meta information and details needed by the UI. The `Question` entity contains a `title` and a `type` (multiple choice or open ended). The `Answer` entity includes the `answer` text and `choice` number (multiple choice answer number). The cardinality between entities is as follows:

- A `PDF` can have zero to many `Questions`.
- A `Question` has at most one correct `Answer`.
- A `Question` may have one or more potential `Answers`.

Inserting data
Core Data automatically generates classes to represent each entity in the data model. Additional methods can be added to these classes using categories[14]. To allow PDFs to be inserted into the model, the `PDFWithPath:` method was implemented within the `PDF+Path` category. This method inserts a PDF into the data model providing it does not already exist (method listed in appendix G.2). Similar categories are defined the for `Question` and `Answer` entities.

[12]SQLlite is a small relational database management system - `http://www.sqlite.org`
[13] A sandbox is a set of fine-grained controls that limit an applications access to files and other resources
[14]Categories provide the ability to add functionality to a class without subclassing or changing the actual object.

Accessing data

The data model is accessed through a `UIManagedDocument` object, which provides mechanisms to request objects from the model. A `UIManagedDocument` variable was added to the `IPMyFilesViewController` to allow it to retrieve `PDF`s from the data model. In order for the view controller to access `PDF` objects, the data model needs to be opened and configured to request the correct data. When the application is launched for the first time, the data model needs to be created as it will not exist. This is achieved by calling the `saveToURL:` method on the `UIManagedDocument`. For all subsequent launches the data model will exist so it can be opened by invoking the `openWithCompletionHandler:` method.

```
- (void) viewWillAppear:(BOOL)animated{ // IPMyFilesViewController
    if (!_pdfDatabase){ // get a reference to the database
        NSURL *url = [[[NSFileManager defaultManager] URLsForDirectory:NSDocumentDirectory inDomains:NSUserDomainMask] lastObject];
        self.pdfDatabase = [[UIManagedDocument alloc] initWithFileURL:[url URLByAppendingPathComponent:@"InteractivePresenter"]];
        [self useDocumet]; // open/load data model
}
- (void) useDocument{ // start using document db
    if (![[NSFileManager defaultManager] fileExistsAtPath :[_pdfDatabase.fileURL path ]]) { // case 1: data model does not exist
        [_pdfDatabase saveToURL:_pdfDatabase.fileURL forSaveOperation:UIDocumentSaveForCreating completionHandler:^(BOOL success){
            [self fetchPDFsIntoDocument:_pdfDatabase]; // fetch PDFs in device storage
            [self setupFetchedResultsController ]; // link table to datasource
        }];
    } else if (_pdfDatabase.documentState == UIDocumentStateClosed){ //case 2: data model exists but it's closed
        [_pdfDatabase openWithCompletionHandler:^(BOOL success) {
            [self setupFetchedResultsController ]; }]; // link table to datasource
}
```

Once the data model is open, objects can be retrieved using a `NSFetchRequest`, which defines what data is requested from the database. The `IPMyFilesViewController` creates a request which states that data should be extracted in ascending order from the `PDF` entity. This request is passed to a `NSFetchedResultsController`, which efficiently manages the results returned from the Core Data model.

```
- (void) setupFetchedResultsController { // IPMyFilesViewController
    NSFetchRequest *request = [NSFetchRequest fetchRequestWithEntityName:@"PDF"];
    request.sortDescriptors = [NSArray arrayWithObject:[NSSortDescriptor sortDescriptorWithKey:@"title" ascending:YES selector:@selector(localizedCaseCompare
        :)]];
    self.fetchedResultsController = [[NSFetchedResultsController alloc] initWithFetchRequest:request managedObjectContext:_pdfDatabase.managedObjectContext
        sectionNameKeyPath:nil cacheName:nil];
```

The `NSFetchedResultsController` is then used to determine the number of cells in the grid and configure their views.

```
- (NSInteger)numberOfItemsInGMGridView:(GMGridView *)gridView{
    id <NSFetchedResultsSectionInfo> sectionInfo = [[self.fetchedResultsController sections] objectAtIndex :0];
    return [sectionInfo numberOfObjects];
}
- (GMGridViewCell *)GMGridView:(GMGridView *)gridView cellForItemAtIndex:(indexPath)indexPath{
    PDFGridCell *cell = (PDFGridCell *)[gridView dequeueReusableCell];  // get cell view to configure
    PDF *pdf = [self.fetchedResultsController objectAtIndexPath :indexPath ]; // ask NSFRC for the PDF for this row
    ... // configure and return cell
}
```

4.1.4 Document Importing

Cloud Storage

There are several popular cloud storage providers which offer personal plans, including Dropbox[15], Google Drive[16] and Box[17]. Each service has its own API to allow developers to integrate the service into their applications. As many developers wish to integrate multiple cloud services into their applications, there are several third party frameworks which simplify the integration process. In this application, the FilePicker[18] framework is used to allow users to import PDFs from Dropbox, Google Drive, Box, Gmail and GitHub. The framework

[15] Dropbox Cloud storage service - `https://www.dropbox.com`

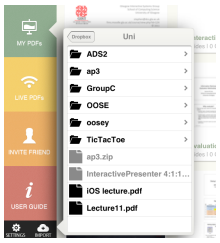
[16] Google Drive - `https://www.google.com/intl/en_GB/drive/start/index.html`

[17] Box - `https://www.box.com`

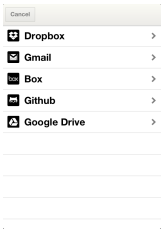
[18] FilePicker - `https://developers.filepicker.io/docs/ios/`

includes an import view controller (`FPPickerController`), which allows users to sign-in to their cloud storage accounts and access their files. The view controller is initialised, configured and presented to the user when they tap the 'import' button. Once the view controller has been initialised, the supported data type is set to PDF and the delegate is assigned to the presenting view controller (`IPMyFilesViewController`). When a user selects a file the `didFinishPickingMediaWithInfo:` method is invoked on the delegate, providing the path to the downloaded PDF. This file is then moved into the documents directory and added to the data model, making it available for selection in the `IPMyFilesViewController` grid view.

```
- (void) FPPickerController :( FPPickerController *)picker didFinishPickingMediaWithInfo :( NSDictionary *)info { // FPPickerControllerDelegate method
    NSURL *tempFilePath = [info objectForKey:@"FPPickerControllerMediaURL"];
    NSString *savedURL = [BCUtility movePDFintoApp:tempFilePath]; // utility method to move pdf into documents directory and add to database
    [ self addPDFAtPath:savedURL]; // add PDF to grid view
    [ picker dismissModalViewControllerAnimated:YES]; // dismiss the filepicker control
}
```



(a) FilePicker presented in a popover on iPad



(b) FilePicker presented modally on iPhone

Figure 4.10: PDF import from cloud storage service using FilePicker

To make best use of the screen real-estate available, the application presents the FilePicker differently depending on the device. On the iPad the picker is layered on top of the existing view in a small popover (figure 4.10a). On iPhone the picker is presented in a full screen modal view, covering the existing view (figure 4.10b).

iTunes File Sharing

Unlike other USB storage devices, iOS devices are not mounted as as standard disk drive when connected to a computer. However, applications can support iTunes File Sharing[19], which allows users to transfer files into the application from their computer using iTunes. This is achieved by adding a key to the application's information property list file (Info.plist)[20].

```
    <key>UIFileSharingEnabled</key> <true/>
```

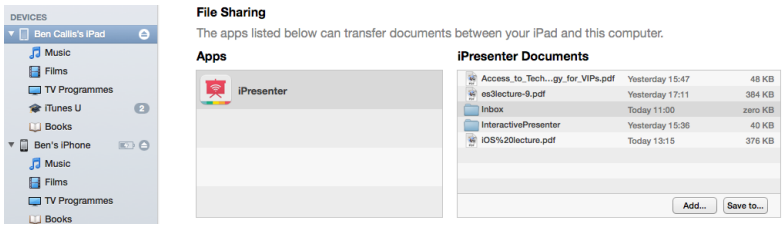


Figure 4.11: iTunes file sharing

Once this has been declared, the application appears in the *File Sharing* list, illustrated in figure 4.11, allowing files to be added to and copied from the applications *Documents* directory. It is not possible to declare what files can be transferred, so users can add files of any type into the application. To address this issue, a method was created which returns an array of paths to PDF files in the the `Documents` directory, ignoring other file types. This method is invoked when the application is launched and the resulting array is traversed to add any new PDFs into the data model. As there could be several PDFs in the directory, this is asynchronously in a separate thread to keep the UI responsive.

[19] iTunes File Sharing - `http://support.apple.com/kb/HT4094`

[20] The Info.plist file is a structured text file that contains essential configuration information for a bundled executable

```
- (void) fetchPDFsIntoDocument: (UIManagedDocument *)document{ // adds new PDFs on device into database
    dispatch_async ( dispatch_queue_create ("PDF_fetcher", NULL), ^{ // could be slow if there is LOADS of docs so do in another thread
        NSArray *pdfs = [ self fetchPDFsOnDevice]; // gets PDFs on the device from iTunes File Sharing
        [document.managedObjectContext performBlock:^{ // add pdfs to CoreData db. Needs to be done on the documents context thread .
            for (NSString *pdfPath in pdfs)
                [PDF PDFWithStringPath:pdfPath inManagedObjectContext:document.managedObjectContext]; // create and add pdf to db. If exists ignore
```

4.1.5 Networking

All iOS devices have Wi-Fi and bluetooth capabilities, which applications can interact with via the iOS SDK. There are advantages and disadvantages to each wireless technology. For example, Wi-Fi offers speeds of up to 54Mbps (Wireless G), significantly faster than Bluetooth (2.0) which has a maximum data transfer rate of 3Mbps. Wi-Fi networks also have a larger range of up to 100 metres, compared to Bluetooth's range of approximately 10 metres. Wi-Fi requires additional hardware to transmit messages between iOS devices, such as a router or access point. Whereas Bluetooth allows devices to communicate without any additional hardware, making it possible to transmit data in any location.
As networking is fundamental to the application, a considerable amount of time was spent researching different approaches, which are described below.

Game Kit is an iOS framework created by Apple and first introduced in iOS 3. The Game Kit framework contains APIs to allow seamless communication over a Wi-Fi or Bluetooth network. The framework aims to make cross device communication simple, with a minimal set of interfaces; providing a high level of abstraction over the Bluetooth and Wi-Fi networking layers. This high level of abstraction allows developers to treat Bluetooth and Wi-Fi as one big network; significantly reducing the amount of networking code required. The framework includes methods to discover other devices on the network and supports client-server and peer-to-peer topologies (discussed in section 3.3.2). It was introduced to allow developers to create local multiplayer games. The framework allows data to be exchanged between devices in packets (formatted unit of data, sent over a network) of up to 87 kilobytes. This packet size is adequate for multiplayer gaming applications, as they are only required to send small updates, such as player location, between devices. However, PDF files are often several megabytes (Mb), significantly larger than the maximum packet size supported by Game Kit.

CocoaAsyncSocket is an open source networking library for iOS and Mac OSX. The library is essentially a wrapper[21] that sits on top of the low level Berkley Socket API[22], removing the need to use standard C functions, leading to a much easier to use networking framework. The library is frequently updated and has extensive documentation. Although the library makes networking easier, it does not offer the same level of abstraction as Game Kit. For example, for a device to receive data over the network it specifically needs to invoke a read method. In Game Kit when a message is sent, the recipient automatically receives it and invokes a method stating what has been received. CocoaAsyncSocket does not provide a method to discover other devices, so another piece of software would be needed in conjunction with this, such as Bonjour (zero configuration networking) [23].

Due to the high level abstraction provided by Game Kit, it was chosen as the networking framework. However, to integrate the framework into the application several modifications were made to improve the connect process and allow PDFs to be distributed between devices.

Establishing a connection

Before any data can be exchanged over Game Kit a connection needs to be made. Game Kit offers a standard user interface for the discovery and connection process (illustrated in figure 4.12). However, this interface requires the server to manually search and accept clients, which would not be practical during a presentation, considering viewers could arrive at different times. To allow documents to be presented with little or no planning

[21] a thin layer of code which adapts a library's existing interface into a compatible interface

[22] The Berkeley sockets API comprises a library for developing applications in the C programming language that perform inter-process communication across a network

[23] Bonjour enables automatic discovery of devices on a local network - `https://developer.apple.com/bonjour/`

(ad hoc networking) a custom view controller (`LivePDFsVC`) and two additional session managing classes were created: one for the client and one for the server.

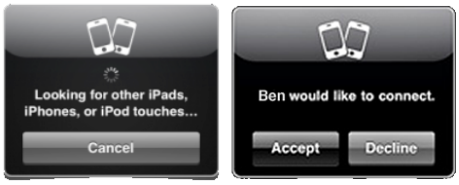


Figure 4.12: Standard Game Kit picker

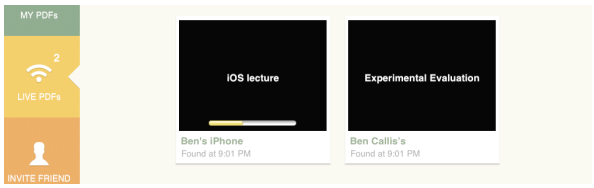


Figure 4.13: Live PDFs VC - Custom picker iPad

Figure 4.13 illustrates the custom view, which is controlled by the `LivePDFsVC`. This view lists all the available presentations in a `GMGridView` and allows users to connect to any live presentation by tapping a cell. The grid is automatically updated when the availability of a presentation changes.

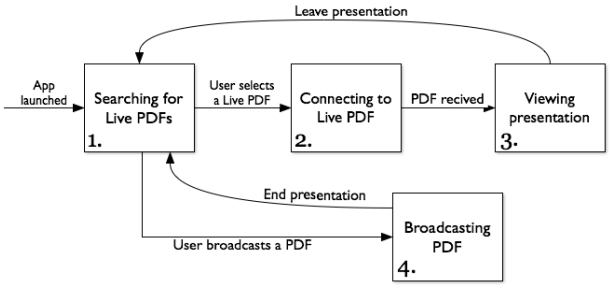


Figure 4.14: Networking States

Figure 4.14 gives a visual representation of the networking states (number 1-4) of the application. When the application is launched the client session manager `IPSearchingPresentationClient` is created, which automatically starts searching for any servers that are broadcasting a presentation (state 1). At this point a user can either connect to a Live PDF, or become a server by presenting one of their own PDFs. If a user selects a Live PDF from the `IPLivePDFsVC`, a connection request is sent the the sever (state 2). When a connection is established, the PDF is transmitted to the viewer (discussed later in this section) and opened in the document viewer (state 3). At this point, the device stops searching, as a connection has been established with the server. In this state, the user receives actions from the presenter which keep the screen synchronised (presenter action re discussed later in this section). When the user leaves the presentation, the application is returned to the original state where it searches for servers (state 1).
Alternatively, a user can become a presenter, by selecting a PDF from the 'My PDFs' view and selecting the broadcast button located in the top toolbar (shown previously in figure in 4.6). When a user broadcasts a PDF, an instance of the `IPBroadcast` is created to handle networking communication. At this point the PDF becomes visible to other devices in the `IPLivePDFsVC`.

The `IPBroadcast` maintains a list of connected clients and stores a `GKSession` object, which is used to transfer data across the network. It interacts directly with the document viewer, allowing presenter actions to be distributed to the connected clients. The class also conforms to the `GKSessionDelegate` protocol to receive important network events and acts as the data handler to receive incoming data. When the class is initialised, a `GKsession` instance is allocated in server mode.

```
1        _session = [[GKSession alloc] initWithSessionID:kSessionID displayName:escpaedInfoString sessionMode:GKSessionModeServer];
2        _session.delegate = self;
3        _session.available = YES;
4        [_session setDataReceiveHandler:self withContext:nil]
```

Setting the `sessionMode` to server instructs Game Kit to broadcast the availability of the service. The `sessionID` is a unique string which allows clients to identify relevant servers. The application uses the string *INTERACTIVE-PRESENTER* which is defined in the constant file. The `GKSession` object also allows a user-

readable name to be set, allowing viewers to identify servers. Initially this name was configured to the presentation name. However, as there may be times when a presentation can not be identified from its name alone, due to multiple broadcasts with the same name or bad file naming, the presenter's name is concatenated onto the end of the presentation name. In order for the client devices to determine when the presentation name ends, the strings are separated by '**' delimiter, and all '*' characters in the strings are escaped with a '/' character. This was implemented by adding the following two class methods to the `BCUtility` class.

```
+ (NSString *) escapedStringWithPresenterName:(NSString *)name PDTitle:(NSString *) title { // performed by server
    NSString *escapedName = [name stringByReplacingOccurrencesOfString:@"*" withString:@"/*"]; // if any * in name replace with /*
    NSString *escapedTitle = [[ title stringByReplacingOccurrencesOfString:@"*" withString:@"/*"] stringByDeletingPathExtension ]; // pdf title
    return [NSString stringWithFormat:@"%@**%@", escapedTitle, escapedName]; // title**name
}
+ (NSDictionary*) pdfInfoDictionaryFromEscapedString:(NSString *)str { // performed by client
    NSArray *parts = [ str componentsSeparatedByString:@"**"]; // split the string into title and name
     ...
```

When the server receives a connection request, the `GKSession` invokes the `didReceiveConnectionRequestFromPeer:` method on its delegate (`IPBroadcast`). This method is configured to automatically accept all incoming connection requests without prompting the presenter for confirmation.

```
- (void)session:(GKSession *)session didReceiveConnectionRequestFromPeer:(NSString *)peerID{
    [session acceptConnectionFromPeer:peerID error:nil]; // accept all viewers. Once accepted the file transfer can begin
}
- (void)session:(GKSession *)session peer:(NSString *)peerID didChangeState:(GKPeerConnectionState)state{
    switch (state){
        case GKPeerStateConnected: [self sendPDFtoPeer:peerID]; // send PDF to viewer
        case GKPeerStateDisconnected: [_readyViewersDict removeObjectForKey:peerID]; // viewer left remove from list of viewers
```

Once the request has been accepted, the `session:peer:didChangeState:` method is called with a state value of `GKPeerStateConnected`, which starts the PDF transfer (discussed in section 4.1.5).

The `IPSearchingPresentationClient` class acts as a mediator between a `GKSession` object and the `IPLivePDFsVC`, shown in figure 4.13. It stores the session, a list of available servers and defines the `IPSearchingPresentationDelegate` protocol, which is used to interact with the `IPLivePDFsVC`. The class conforms to the `GKSessionDelegate` protocol and acts as the data handler for the session. When the application is launched, an instance of this class is created, which instantiates a `GKSession` in client mode and starts searching for available serves with the same sessionID.

```
        _session = [[GKSession alloc] initWithSessionID:sessionID displayName:nil sessionMode:GKSessionModeClient];
```

When devices on the network change state, the `IPSearchingPresentationClient` is informed via the `session:peer:didChangeState:` delegate method. This method includes a state parameter and a peerID, describing a device's status. There are five states in total; however, the most important states are `GKPeerStateAvailable` and `GKPeerStateUnavailable`. The available state indicates that a new presentation has become available, and the unavailable state signals that a presentation has ended. When either of these states occur, the `LivePDFViewController` is informed via a delegate method, so it can change the availability of the presentation accordingly.

```
- (void) session:(GKSession *)session peer:(NSString *)peerID didChangeState:(GKPeerConnectionState)state{ // IPSearchingPresentationClient
    switch (state){
        case GKPeerStateAvailable: // The client has discovered a new server.
            [_availableServers addObject:peerID]; // add to list of available servers
            [self.delegate matchmakingClient:self serverBecameAvailable:peerID]; // inform LivePDFsViewContorller so it can add it to the grid
        case GKPeerStateUnavailable: ... // server gone away. inform the LivePDFsViewContorller so it can remove it from the grid
```

When a user selects a Live PDF from the `IPLivePDFsViewController`, a connection request is sent via the `IPSearchingPresentationClient`, which starts the file transfer process.

```
- (void)GMGridView:(GMGridView *)gridView didTapOnItemAtIndex:(NSInteger)position{
    LivePDFGridCell *cell = (LivePDFGridCell *)[_gmGridView cellForItemAtIndex:position]; // get the actual cell
    NSString *peerID = [ _searchingPresentationClient peerIDForAvailableServerAtIndex:position]; // get peerID from IPSearchingPresentationClient
    [ _searchingPresentationClient connectToServerWithPeerID:peerID]; // connect
      ..
```

Transmitting the PDF

As previously mentioned, Game Kit has a maximum packet size of 87k, which is significantly smaller than the average PDF size. Additional research was carried out to identify different ways to transfer a file between devices at the start of a presentation. The following three approaches were identified:

Cloud Storage As the application already makes use of online cloud storage services, it would be possible to upload and share PDFs using these services. For example, DropBox allows files to be uploaded and shared with other through a public URL. At the start of a broadcast if the file was not already located in cloud storage it would be uploaded. The presenter would then send the URL of the PDF to each viewer once a connection had been established. The `FilePicker` component could be used to do the upload process, so no additional libraries would be necessary. However, an internet connection would be required and all presentations would be made publicly available during the presentation. This approach is used by IdeaFlight[1], which was discussed in section 2.2.2.

Split PDF into packets PDFs could be transmitted solely over Game Kit by splitting the file into multiple packets. However, additional information would need to be included with each packet to ensure that viewers could reconstruct the PDF even if packets got lost, corrupt or arrived out of order. While this approach is complex, it requires no additional frameworks and will work over Bluetooth or Wi-Fi.

HTTP Server The file could be made accessible through a HTTP connection by creating a web server on the presenting device. Once a viewer establishes a connection with a presenter, the URL to the file would be transmitted via Game Kit. There are several open source libraries available to embed HTTP servers into iOS applications, including: CocoaHTTPServer[24] and MongooseDaemon[25]. Using a HTTP server would result in fast direct file transfer without out the need to split the PDF into multiple packets. However, it would increase memory usage and require Wi-Fi.

To support Bluetooth and Wi-Fi, it was decided that PDFs should be split into multiple packets and transferred over Game Kit. This behaviour was implemented in the `sendPDFtoPeer` method, which is invoked when a viewer established a connection to a presenter. This method splits the PDF into multiple packets and transmits each packet reliably (using `GKSendDataReliable`[26]) to the viewer.

```
1 - (BOOL)sendPDFtoPeer:(NSString *)peerID {
2     if (!_pdfNSData) _pdfNSData = [NSData dataWithContentsOfFile: [_document.fileURL path ]]; // convert pdf to NSData
3     NSArray *packets = [ self packets ]; // split pdf data into packets <87k each
4     for(NSData* packet in packets) { // send packets one by one
5         NSError* error = nil;
6         [ _session sendData:packet toPeers :[ NSArray arrayWithObject:peerID] withDataMode:GKSendDataReliable error:&error]; // GKSession method
```

Each packet contains a PDF identifier (UDID), a PDF MD5, the PDF name, PDF data (payload), payload start and payload length. The packets are sent as an `NDData` object which is constructed by serialising a dictionary of key value pairs. A visual representation of a packet is shown in 4.15.

Figure 4.15: PDF packet

When the packets are received, they are reassembled using the additional packet information by the `IPSearching PresentationClient`. This is achieved by constructing a `NSMutableData` object matching the size of the PDF, and replacing ranges of bytes with the data contained in each packet's payload.

```
1 - (void)receiveData :(NSData*)data fromPeer:(NSString*)peer inSession :(GKSession*)s context :(void*)context { // in IPSearchingPresentationClient
```

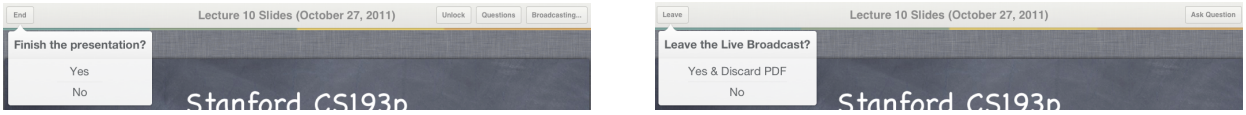
[24] CocoaHTTPServer - `https://github.com/robbiehanson/CocoaHTTPServer`
[25] MongooseDaemon -`https://github.com/face/MongooseDaemon`
[26] In GKSendDataReliable mode a packet is automatically retransmitted if it fails to reach its destination

```
NSDictionary *packet = [ NSPropertyListSerialization propertyListWithData:data options:0 format:NULL error:nil ]; // Extract data into dict
if(received == nil) // if first packet we have received for this file
    NSMutableData* data = [NSMutableData dataWithCapacity:[packet objectForKey:@"PDF_Size"] intValue]];
[[ received objectForKey:@"payload"] replaceBytesInRange:NSMakeRange([[packet objectForKey:@"start"] intValue ],
    [[ packet objectForKey:@"length"] intValue ]) withBytes :[( NSData*)[packet objectForKey:@"payload"] bytes ]]; // replace bytes in the range
... // check if the file has been fully received . If it has notify the LivePDFsViewController
```

Once the PDF has been successfully transmitted the `IPSearchingPresentationClient` informs the `IPLivePDFsViewController`, which opens the PDF using the document viewer and creates an instance of the `IPBroadcast` using the same session.

Presenter Actions

The document viewer's appearance and behaviour is different for viewers and presenters. The top toolbar contains different controls giving each user direct access to the available functionality. For example, the presenter has access to a 'broadcast' button, which is not available to viewers. This is illustrated in figure 4.16.



(a) Presenter View (b) Viewer View

Figure 4.16: Presenters and Viewers have different controls when viewing the same PDF

As a presenter, interaction with the document is monitored. Actions such as page changing, zooming and pointing are forwarded on to all viewers to keep the presentation synchronised. Each action is sent across the network in a single packet. All action packets contain a header, packet number, packet type and playload (illustrated in figure 4.17).

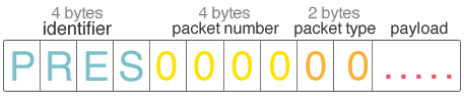


Figure 4.17: 10 byte Packet Header

The first four bytes in the header act as an identifier to determine if the packet relates to the application. Any packets which do not start with this identifier are discarded, to increase the robustness and security of the application. The succeeding four bytes indicate the packet number, which is used to recognise when a packet arrives out-of-order. This allows non-important packets which are received late to be discarded. The final two bytes represent the packet type, which is used to determine the contents of the packet's payload.

To synchronise the presentation across devices four packet types are required: page change, scroll/zoom, laser pointer and client ready. As each packet has certain similarities, an abstract packet class `IPPacket` was created to provide a level of abstraction to handling packets. This class includes methods and variables which are common to all packet types. Each packet type can provide different packet semantics through subclassing the `IPPacket` class. This packet class hierarchy is illustrated in figure 4.18.

The `packetType` variable is of type `packetType` enum. The enum is defined within the `IPPacket`, to give a textual name to each numerical `packetType`. The `IPPacket` class includes a convenience factory method `packetWithData:` which defers instantiation to the appropriate subclass. The class also includes a `data` method which returns an NSData object with the contents of the packet. This method creates a mutable data object and appends the packet header (identifier, packet number and packet type) to it as well as any additional payload data.

```
- (NSData *)data{
    NSMutableData *data = [[NSMutableData alloc] initWithCapacity :100];
    [data rw_appendInt32:'PRES'];    // identifier
    [data rw_appendInt32:0];  // packet number
```

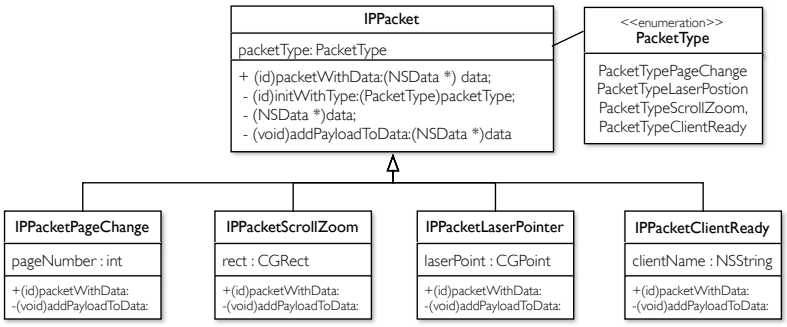


Figure 4.18: Packet classes *(initial 4)*. Using the Factory design pattern.

```
    [data rw_appendInt16:self.packetType]; // packet type used to identify packet when recived
    [self addPayloadToData:data]; // gets the subclasses to add any extra data (e.g page number)
    return data;
```

Each subclass of `IPPacket` includes a custom implementation of `addPayLoadToData` method, which adds the necessary payload to the packet. For example, the `IPPackerLaserPointer` adds the `x` and `y` co-ordinate for the laser pointer to the payload.

```
- (void)addPayloadToData:(NSMutableData *)data{
    [data rw_appendInt32:self.laserPoint.x]; // x co-ordinate of the laser
    [data rw_appendInt32:self.laserPoint.y]; // y co-ordinate of the laser
```

The `rw_append` methods were added to the `NSMutableData` class within the `NSMutableData+NetworkingAddidtons` category.

```
- (void)rw_appendInt16:(short)value{
    value = htons(value); // network byte order aka big endian
    [self appendBytes:&value length:2]; // 2 bytes == 16 bits
```

In the `rw_appendInt16` method (listed above) the contents of `value` are appended to the packet. Before appending the data, `htons` (a C function) is called to ensure the value is in *network byte order*[27]. This increases the robustness of the protocol as it guarantees that all packets generated from different devices are transmitted with the same byte ordering.

An `IPPacketLaserPointer` is created when the presenter performs a long tap gesture. This packet is then sent to all connected viewers, which process the packet and display the laser pointer at the specified point. When a viewer receives a packet it converts the binary data (`NSData`) back into a subclass of IPPacket using the `packetWithData:` factory method.

```
+ (id)packetWithData:(NSData *)data{ // in IPPacket class
    if (([data length] < PACKET_HEADER_SIZE) && (data rw_int32AtOffset:0] != 'PRES')) // check packet length and identifier to see if its valid
        NSLog(@"Error:_Packet_has_invalid_header"); return nil;
    int packetNumber = [data rw_int32AtOffset:4]; // get the packet number
    PacketType packetType = [data rw_int16AtOffset:8]; // get the packet types
    IPPacket *packet;
    switch (packetType){ // remake the packet into its concrete type
        case PacketTypePageChange:
            packet = [IPPacketServerPageChange packetWithData:data]; break; // details of this method below
        case PacketTypeScrollZoom:
            packet = [IPPacketScrollZoom packetWithData:data]; break;
        case PacketTypeLaserPostion:
            packet = [IPPacketLaserPointerPosition packetWithData:data]; break;
        ... // all other packet types
        default:
            NSLog(@"Error:_Packet_has_invalid_type"); // don't crash if receive odd packets. Just ignore them and write to log
    } return packet; // The packet is later cast to the concrete type by checking the packetType variable
}
```

[27] Network Byte Order - canonical byte order convention for data transmitted over the network

Each concrete packet defines its own implementation of the `packetWidthData:` method, which reconstructs the binary payload data into the relevant data types.

```
+ (id)packetWithData:(NSData *)data{ //  PPacketLaserPointer
    size_t  offset  = kPacketHeaderSize; //  start  reading  after  the  header
    int  laserPointX  = [data  rw_int32AtOffset : offset ];  //  convert  the  first  32  bits  of  data  back  to  the  x  co-ordinate
    offset +=4;  //  32  bits  = 4  bytes  so  move the  offset  along
    int  laserPointY  = [data  rw_int32AtOffset : offset ];  //  convert  the  next  32  bits  of  data  back  to  the  y  co-ordinate
    return  [[ self  class ]  packetWithlaserPoint :CGPointMake(laserPointX, laserPointY ) ];  //  return  a  correctly  initialised  PPacketLaserPointer  packet
}
```

Figure 4.19 illustrates when a `IPPackerLaserPointer` packet is created and distributed to viewers.

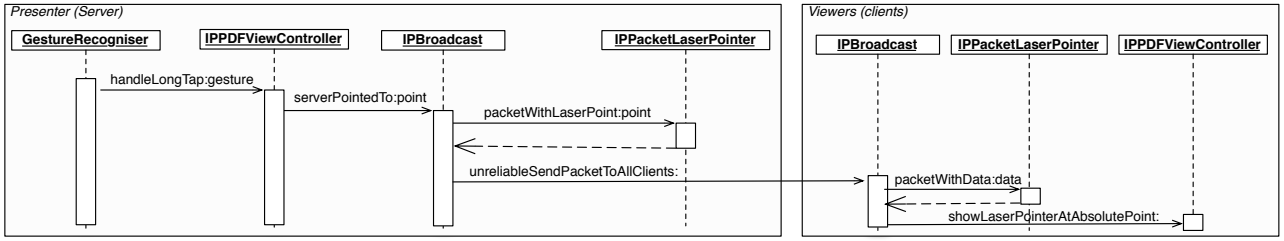
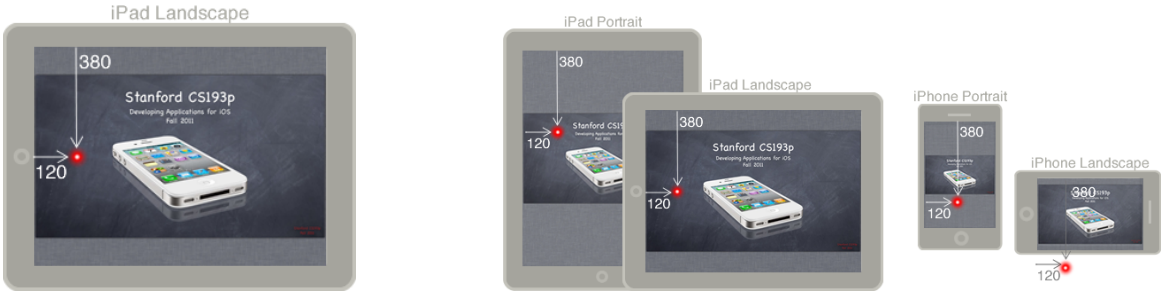


Figure 4.19: Laser pointer packet creation and distribution

The initial implementation of the laser pointer packet included the x and y co-ordinate of the long tap gesture. This approach resulted in positioning problems as it did not take into account that viewers may be using different devices, in multiple orientations. Figure 4.20 shows how the laser pointer appeared on different devices using this approach.



(a) Presenter View

(b) Viewer View

Figure 4.20: Laser point wrongly positioned when using raw touchPoint co-ordinates.

A naive approach to solving this issue would be to send details of the presenters device and orientation to each viewer. This would allow viewers to scale the co-ordinates for their device accordingly. However, the flexibility of this solution is limited as the application would need to be modified if a new iOS device was released.
Instead, the gestures' co-ordinates are converted into the content view of the PDF page before transmission. The PDF content view co-ordinate system is based on the dimensions of the PDF page, which is identical on all devices. Figure 4.21 illustrates the two different view co-ordinate systems. When a viewer receives a laser pointer packet, the co-ordinates are converted to document viewers co-ordinate system before the pointer is displayed.

The initial implementation of `IPPacketScrollZoom` included two values: a centre screen position and zoom level. This approach had similar view issues to the initial laser pointer packet. This was resolved by replacing the two values with a `CGRect`[28] denoting the area of the PDF page currently in view. When this packet is received by viewers, the `zoomToRect:animated:` method is invoked which zooms to the specific

[28]CGRect - a data struct which represents the location and dimensions of a rectangle

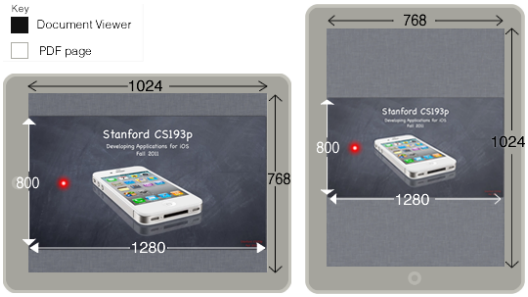


Figure 4.21: View co-ordinate systems

area of the PDF page so the area contained within the packet it is visible. This method is animated to provide viewers with additional context.

An `IPPacketPageChange` is generated every time the presenter switches page in the document. The packet includes an integer representing the current page number, which is used to keep all devices on the same page.

The `IPPacketClientReady` is used to request the current state of the presentation. It is sent by a viewer to the presenter after the PDF transfer has completed, to accommodate viewers who join after the presentation starts. When a presenter receives the packet, it sends the current page and scroll/zoom position.

The majority of the packets in the application are transmitted reliably, as delivery guarantees are required. However, as an `IPPacketLaserPointer` is created every time the laser pointer moves, they are sent unreliably (using `GKSendDataUnreliable`[29]) as delivery speed is more significant than guaranteed delivery to keep the laser pointer movement fluid.

When testing the application over a bluetooth connection, the limited bandwidth resulted in latency issues with the responsiveness of the laser pointer. To improve performance, the `IPPDFViewController` was modified to restrict the number of laser pointer packets generated. This was achieved by only sending laser pointer packets when the laser moved a distance of three or more pixels from the previous point.

```
1  float totalDisMoved = fabs( _lastSentLaserPosition .x - point.x) + fabs( _lastSentLaserPosition .y - point.y) ; // get distance between current point and
       previous point
2  if (totalDisMoved >3){ // restrict number of packets sent
3      [ _broadcast serverPointedToPoint : relativePoint ]; // send to viewers
```

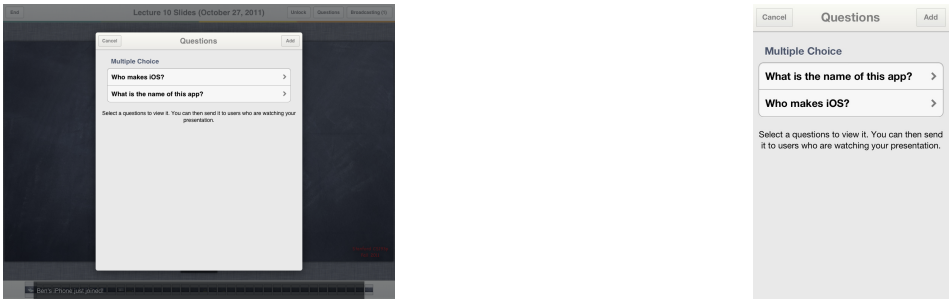
To keep laser pointer transitions smooth, the movement of the laser pointer is animated on the viewer's device.

```
1  [UIView animateWithDuration:0.04 animations:^{ [ self .laserPointerImageView setFrame:newFrame]; }]; // animate laser move on viewers devices
```

4.1.6 Questions and Voting

To allow users to add questions, multiple view controllers were created which interact with the `Question` and `Answer` entities in the data model (discussed in section 4.1.3). Users can add questions to a PDF before or during a presentation by selecting the 'Questions' button in the document viewer toolbar. This presents the `IPQuestionListViewController`, which lists existing questions, provides the ability to select a question to view detailed information and allows new questions to be added. Figure 4.22 shows the view controller presented on the iPhone and iPad. On the iPad, the view is displayed in the centre of the document viewer, allowing the user to observe changes to the document in the background. Due to the limited screen size on the iPhone, the view is displayed full screen on this device.

[29] In GKSendDataUnreliable mode data is sent once so delivery is not guaranteed



(a) iPad (b) iPhone

Figure 4.22: Question list presented on iPad and iPhone

Users can select the 'Add' button to create a new question, which instantiates the `IPAddViewQuestion ViewController` (shown in figure 4.23).

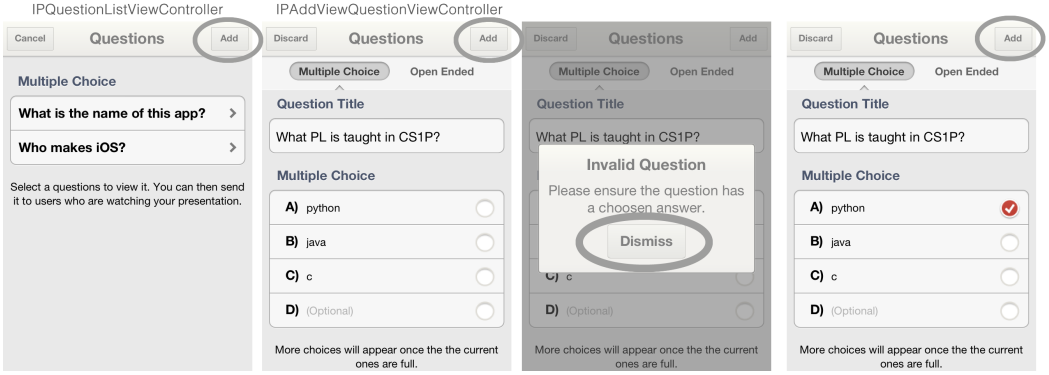


Figure 4.23: Add question

This view includes a custom segmented control, `SDSegmentedControl`[30], which allows the question type to be set. Currently the control includes two types: *multiple choice* and *open ended*. However, more question types can be added to this control in the future. The view controller also includes a `UITableView` which loads custom cells to represent the question and possible answers. The custom cells are of type `IPAnswerCell` and contain a choice label, a textfield to allow user input and a button used to mark the answer as correct. Figure 4.24 illustrates XIB interface file for the (`IPAnswerCell`).

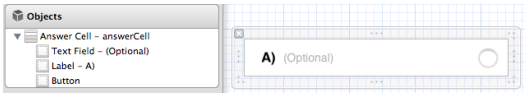


Figure 4.24: Multiple choice custom answer cell

The tableview initially presents two answer cells to inform the user that only two choices are mandatory. The `IPAddViewQuestionViewController` is the delegate for each text field in the `IPAnswerCell` cells, so it is informed via the `textFieldDidEndEditing` method when a user edits any text in the cells. When this method is invoked the content of the cell's text field is checked to determine if a new cell needs to be added as all current cells are full.

```
1 - (void)textFieldDidEndEditing:(UITextField *)textField {
2     NSString *text = textField.text;
3     if (text.length>0){ // if the answer is not blank
4         [_answersArray setObject:text atIndexedSubscript:textField.tag]; // answers stored in an array.
5         if ((tag +1 )== _answersArray.count && _answersArray.count <8){ // add a new if all current ones are full - MAX 8
6             [_answersArray addObject:@""];
```

[30] SDSegmentedControl - `https://github.com/rs/SDSegmentedControl`

```
7                [ self .tableView  reloadData ];
```

The view controller also validates the question before adding it to the data model. If a question is invalid the user is informed by a customised alert (`BlockAlertView`[31]) containing details of the error. An example of the customised `BlockAlertView` is shown in figure 4.23.

To allow questions and answers to be sent over the network, additional packet types were created. The packets were integrated into the `IPPacket` factory design pattern by subclassing `IPPacket` and implementing the required methods. Figure 4.25 shows the four additional packets added to support questions and answers.

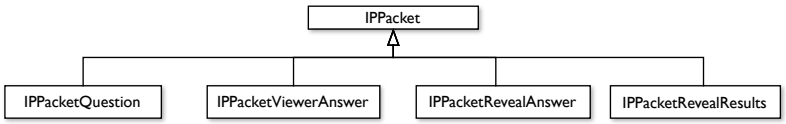


Figure 4.25: Four additional packet types defined for transmitting questions and answers

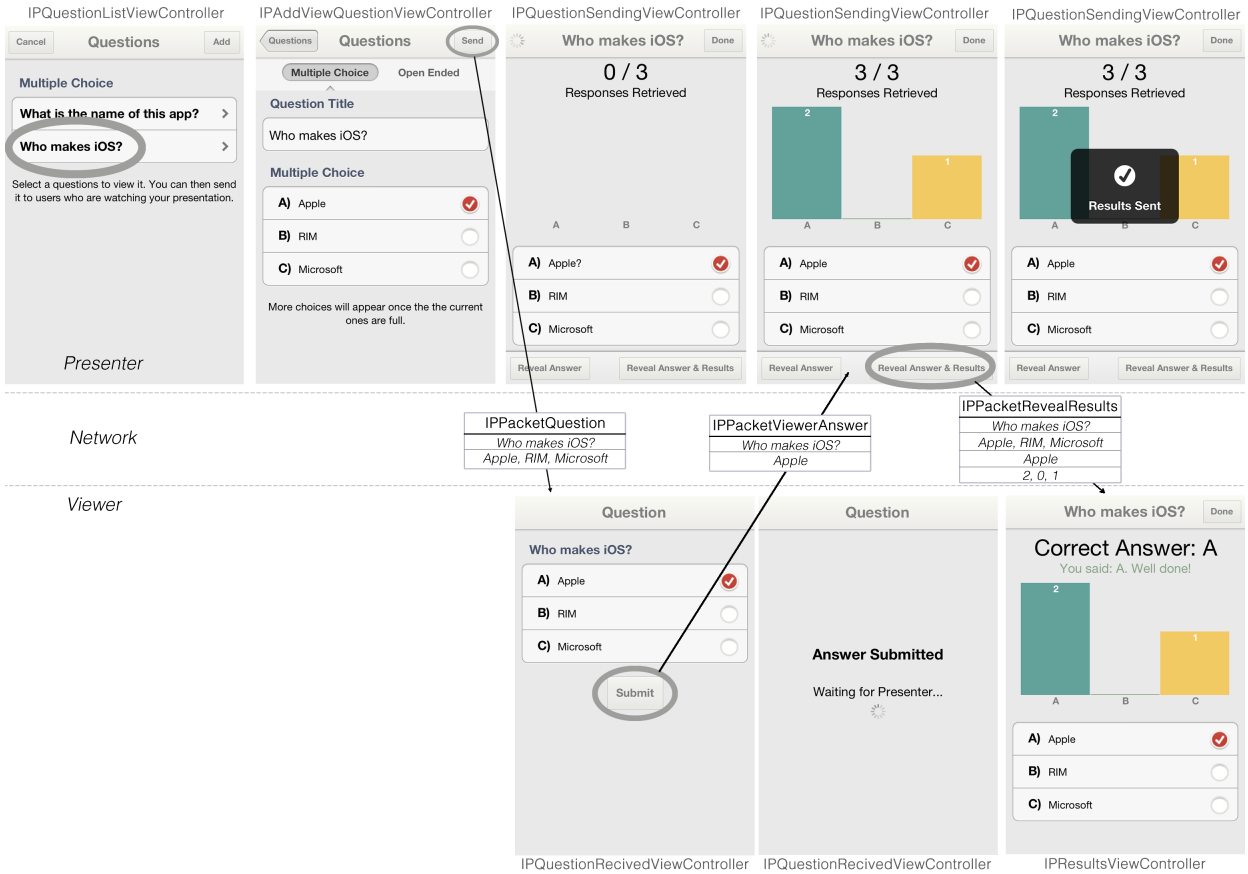


Figure 4.26: Questions and Answer device interaction

Figure 4.26 illustrates the creation and flow of these packets. An `IPPacketQuestion` packet is created when a presenter selects the 'Send' button from the `IPAddViewQuestionViewController`. This packet includes a string representing the question and an array of possible answers. When a viewer receives a packet of this type the `IPQuestionReceivedViewController` is presented, displaying the question to the user. Once an answer has been selected, it is transmitted to the presenter using an `IPPacketViewerAnswer` packet, which contains the question and selected answer. The presenter can observe viewers responses visually within the `IPQuestionSendingViewController`, which is automatically displayed when a question is sent to viewers. The presenter can inform all viewers of the correct answer by selecting the 'Reveal Answer' button

[31] BlockAlertView - `https://github.com/gpambrozio/BlockAlertsAnd-ActionSheets`

situated in the bottom toolbar. This generates and transmits an `IPPacketRevealAnswer` packet containing the question, choices and correct answer. The packet includes the question and possible answers, allowing the results to be displayed to recently joined viewers who did not receive the question. The results can also be transmitted to viewers, allowing the graph to be displayed on their device. This is achieved by selecting the 'Reveal Answer & Results' button located in the bottom toolbar. This generates an `IPPacketRevealResults` packet, which includes all the details contained in the `IPPacketRevealAnswer` and the vote counts for each multiple choice answer.
The results are displayed in a custom bar chart (`BCBarChart`) created specifically for this application. This view is a subclass of `UIView`[32], which overrides the `drawRect:` method to programatically render the chart at runtime. Each time a new response is received the graph is refreshed to update its content. This is animated to provide a fluid visual transition between the graphs states. The view automatically resizes the bars to fit within the view's frame. The value and colour for each bar is stored within an array, which can easily be set by other objects. The drawing code for this view is listed in appendix G.3.

4.2 User Feedback and Refinement

4.2.1 Evaluation Design

After implementing all the 'must have' and 'should have' requirements during the initial development phase, a small scale evaluation was performed. This evaluation was carried out to assess the networking performance when broadcasting to multiple device and to further prioritise the unimplemented 'could have' and 'would be nice to have' requirements (outlined in chapter 3). The evaluation compromised of three parts:

1. Participants were first introduced to the application by viewing a presentation on their device led by a demonstrator. This illustrated all the functionality currently implemented in the application.
2. Participants were then encouraged to test the application in pairs, taking on the role as a presenter and viewer.
3. Users then completed a short questionnaire, which captured usability feedback and ranked additional feature requests.

4.2.2 Results

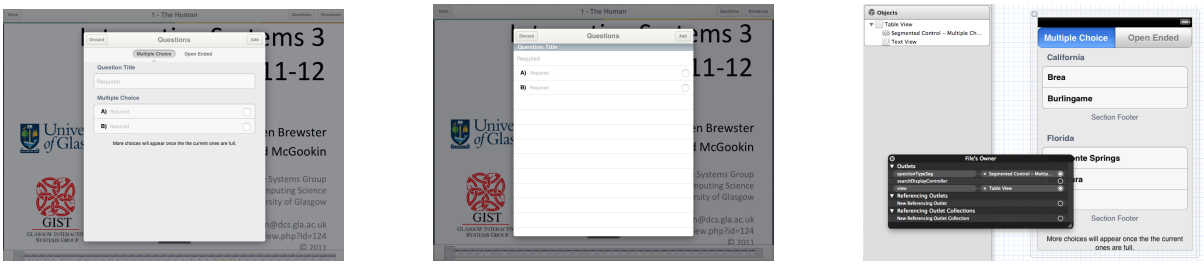
Seven iOS users took part in the evaluation: five university students and two employees currently working in the private sector. The application was tested on iPads, iPhones and iPod Touches running different versions of iOS.

Defects Found
No device had any significant problems, however two defects were identified: firstly, the seventh connected viewer's device sporadically lost connection during a presentation; and secondly, there was an issue with loading customised table views on iPads running iOS 5 (illustrated in figure 4.27). Solutions for both of these defects are described in section 4.3.1.

Usability issues
Many participants had difficulty using the laser pointer, as they were unaware of the required gesture. A user guide will be implemented in the second version which includes a list of presenter controls.

One iPad user also tried to select the *'INTERACTIVE PRESENTER'* logo in the custom navigation bar, as it looked similar to the other tab buttons. The user stated they expected the logo to take them to the applications 'home' screen as common in other applications, such as websites. This behaviour will be implemented during the next implementation cycle.

[32]The UIView class defines a rectangular area on the screen and the interfaces for managing the content in that area.



(a) iOS 6 (b) iOS 5 (c) Interface file (XIB)

Figure 4.27: Customised table view displaying incorrectly on the iPad running iOS 5

Questionnaire results

The questionnaire included a series of closed format question as well as a few open ended questions to encourage participants to give detailed feedback. The most significant results are listed below. Additionally, a complete listing of questions and results can be found in appendix F.

All participants rated the applications interface as Good or Excellent. *"felt like a really nice hybrid functional presentation tool and magazine app. A+ job."* One user praised the animations throughout the application and suggested situations where additional animations would be beneficial. *"Some more animation effects when a peer device connects to share its presentations."*

When asked to state their favourite feature in the application the majority of participants choose the question functionality.*"Question feature, removes the stigma from presentations."*

One of the closed questions required participants to rank the the unimplemented functional requirements by importance. The results are listed in figure 4.28.

Answer Options	1 - High Priority	2	3	4	5	6	7	8	9	10 - Low Priority	Rating Average
Viewer raised questions	1	2	3	1	0	0	0	0	0	0	2.57
Save documents from a presentation	2	2	1	0	1	0	0	1	0	0	3.14
'Open In' support	2	1	0	0	2	0	1	1	0	0	4.14
Lock/Unlock presentation	0	1	2	1	0	0	2	0	0	1	5.14
Open ended questions	0	0	0	4	1	1	0	0	1	0	5.14
Passcode secured presentations	1	0	0	0	2	2	1	0	0	1	5.71
Annotation support	0	1	0	1	0	2	1	1	1	0	6.00
Document Syncing	1	0	0	0	1	0	1	2	1	1	6.86
Note taking via keyboard	0	0	0	0	0	1	1	2	2	0	8.00
Custom usernames	0	0	1	0	0	1	0	0	1	2	8.29

Figure 4.28: User additional feature rank

Participants ranked *Viewer raised questions* as the highest priority, with a rating average of 2.57. *Save documents from a presentation* and *'Open in' support* were also ranked with significant importance. *Lock/unlock presentation*, *Open ended questions* and *Passcode secured presentation* were all ranked with a medium priority (rating average 5-6). However, participants in employment ranked *Passcode secured presentation* significantly higher than students, with one participant marking it as the top priority.

The survey also allowed users to suggest any additional features which they felt would improve application.
"Have some analytics features so admins can see who has viewed and saved the slides - good for attendance"
"Could it be possible for the viewers to also highlight things if they were asking a question."
"Lecturers tend to ask for questions at the end of presentation. Most of the time no questions are asked. Allowing users to send questions in from the application is likely to improve the number of questions raised especially if they can be anonymous"

4.2.3 Additional features to implement

Based on the results from the evaluation the following features were prioritised for the second development cycle:

- Ask the presenter questions (FR15)
- Ability to ask anonymous questions
- Viewer list
- Save document from a presentation (FR9)
- Passcode secure presentation (FR14)
- Navigation lock\unlock (FR8)
- 'Open in' support (FR11)
- User guide
- Additional Animations

4.3 Second Implementation cycle

The second implementation phase aimed to address the defects identified during the initial evaluation, and implement the nine features prioritised by users.

4.3.1 Addressing identified defects

Networking defect
After the evaluation it became apparent that other iOS developers using the GameKit framework had experienced connection dropouts when connecting multiple devices running iOS 5 and iOS 6[33]. This issue only occurs when a `GKSession` configured as a server interacts with a mix of iOS 5 and iOS 6 clients. The issue does not arise if the server is running iOS 5 or all devices are running iOS 6. There is currently no known fix for this issues, however the author and several other developers have informed Apple about this issue in the hope that it will be rectified in a future software update.

Table view defect
The issue loading customised table views on iOS 5 related to the way the view was initialised. When a view controller is allocated using its designated initialiser[34], it automatically inflates a XIB file matching the class name. This was working as expected on iOS 6; however, on iOS 5 the `UITableViewController` initialiser was not locating the XIB file automatically. To resolve this issue, all subclasses of `UITableViewController` were modified to override the designated initialiser and explicitly inflate the interface file.

```
1 - (id) initWithStyle :(UITableViewStyle)style { // IPVAddViewQuestionViewController - subclass of UITableViewController
2     self = [super initWithNibName:@"IPVAddViewQuestionViewController" bundle:nil];
```

4.3.2 Passcode

To implement passcode secure presentations an additional view controller was created to allow broadcast options to be set (`IPBroadcastSetupViewController`). This view allows users to set a passcode and state if viewers can save the PDF at the end of a presentation (discussed in section 4.3.7). Users can enable the passcode security by interacting with a `UISwitch`. When the switch's state is set to true, two textfields are added to the view in an animation block, allowing a user to define a passcode. Users are required to confirm their specified passcode to ensure it is valid. Figure 4.29 illustrates this view on the iPad and iPhone. In order to allow the passcode to be transmitted and verified over the network, two additional packets were created: `IPPacketPasscodeRequest` and `IPPacketPasscodeVerified`. An `IPPacketPasscodeRequest` packet is created when a presenter broadcasts a presentation with a passcode. The packet includes a hashed version of the passcode (produced using the SHA-256 cryptographic hash function[35]) to ensure it can't be identified by an external packet analyser[36]. The `IPBroadcast` class was modified, to send this packet when a connection is established between a viewer and presenter. This packet is received by the `IPSearchingPresentation Client`, which invokes the `receiveData:` method. Additional logic was added to this method to check the

[33] Game Kit bug identified on the Apple Developer Forums - `https://devforums.apple.com/message/764265#764265`
[34] The initialiser of a class that takes the full complement of initialisation parameters.
[35] SHA-2 cryptographic functions - `http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf`
[36] Software or hardware that can intercept and log traffic passing over a network

(a) Passcode disabled by default (b) Passcode enabled (c) Passcode validation

Figure 4.29: Broadcast options View Controller presented on iPad (*left*) and iPhone (*right*)

packet type of the received data (shown below). This enables the class to prompt the user for a passcode when receiving an `IPPacketPasscodeRequest` packet, whilst maintaining the ability to assemble PDF documents from other packets. Figure 4.31 illustrates the passcode prompt which is invoked within the `receiveData` method.

```
1 -(void)receiveData:(NSData*)data fromPeer:(NSString*)peer inSession:(GKSession*)s context:(void*)context{
2     IPPacket *ipPacket = [IPPacket packetWithData:data]; // turn the data into a packet
3     if (ipPacket.packetType == PacketTypePasscodeRequest) {// check if its a passcode request
4         IPPacketPasscodeRequest *passcodePacket = (IPPacketPasscodeRequest *) ipPacket; // cast to passcode packet so can access hashed passcode
5         [self displayPasscodePromptForPasscode:passcodePacket.hashedPass peer:peer inSession:s]; // validate then request PDF when validated
6     }else{ // must be PDF packet ...
```

The passcode input by the user is then hashed with the same cryptographic function before being compared to the actual passcode received from the server. If the viewer inputs the passcode incorrectly, they are informed and prompted to try again. Once the passcode has successfully been verified, an `IPPacketPasscodeVerified` packet is sent to the presenter, which starts the document transfer process. Figure 4.30 provides a visual representation of this behaviour.

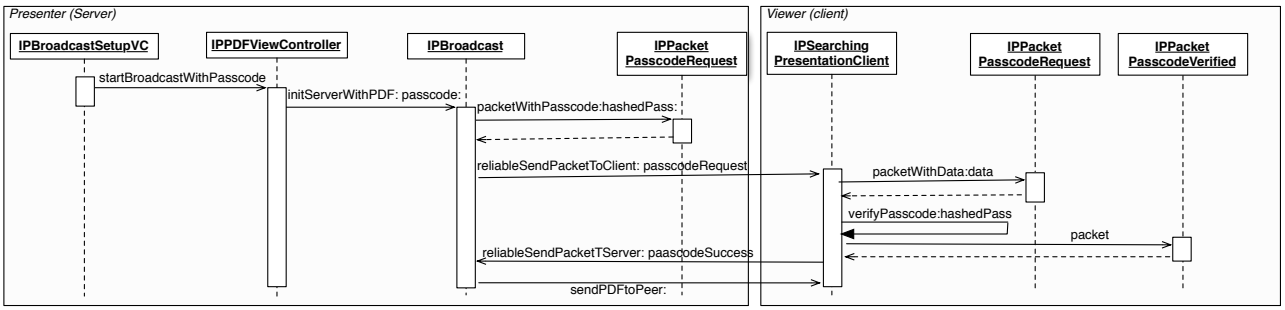
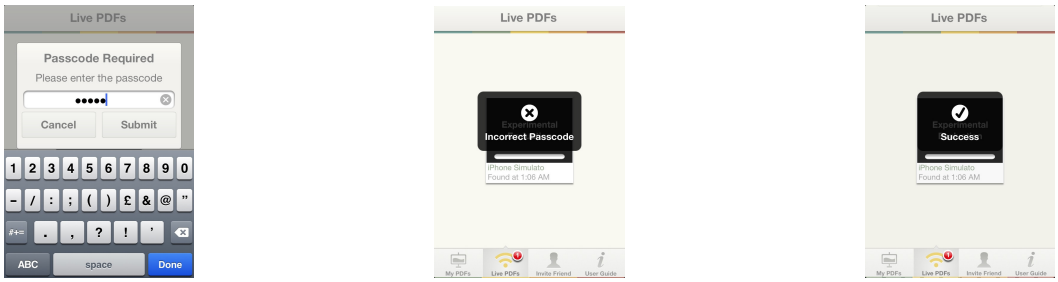


Figure 4.30: Passcode verification between devices



(a) Viewer passcode prompt (b) Invalid passcode alert (c) Passcode verified successully

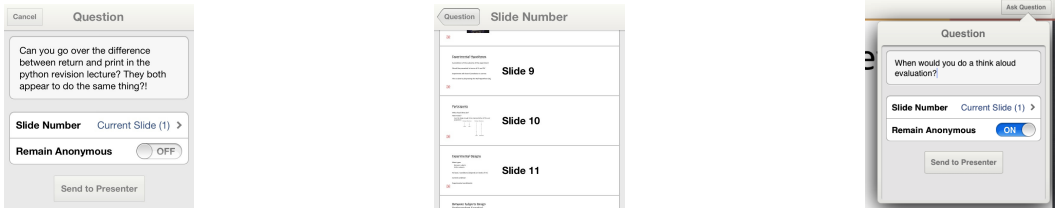
Figure 4.31: Passcode prompt when joining a secured presentation

4.3.3 Viewer List

During the evaluation, one user suggested the presenter should be able to see a list of viewers. To implement this an additional view controller was created (`IPBroadcastViewersVC`), which includes a table of connected viewers. This table is populated by the array of viewers stored in the `IPBroadcast`. The presenter can access this list by tapping the 'Broadcasting' button during a presentation. Figure 4.33 illustrates the viewer list displayed in a popover on the iPad.

4.3.4 Viewer Questions

To allow viewers to create and send questions to the presenter, an additional view controller (`IPUserRaisedQuestionViewController`) was created. Viewers can access this view by selecting the 'Ask Question' button located in the upper right of the top toolbar. Figure 4.32 illustrates this view presented on the iPhone and iPad.



(a) iPhone (b) Ability to include slide number (c) iPad

Figure 4.32: View to allow users to send questions to the presenter

Viewers are able to include a slide number with their question, and indicate if they would like the question to be sent anonymously. When a user selects the 'Send to Presenter' button the view controller informs the `IPPDFViewController`, which subsequently notifies the `IPBroadcast` object. The `IPBroadcast` then creates an `IPPacketUserQuestion` packet and transmits it reliably to the presenter. The `IPPacketUserQuestion` class inherits from the `IPPacket` class and implements the required methods. The `addPayloadToData:` method appends the question title, slide number and anonymous state.

```
1 - (void)addPayloadToData:(NSMutableData *)data{ //  IPPacketUserQuestion
2     [data rw_appendString: _question.question ];        // question
3     [data rw_appendInt32: _question.slideNum];        // page
4     [data rw_appendInt8: ( _question.annon ? '1' : '0' ]; // anonymous state
5 }
```

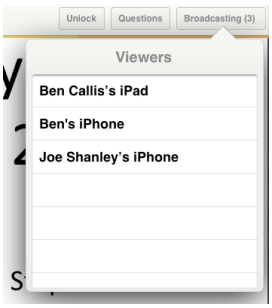
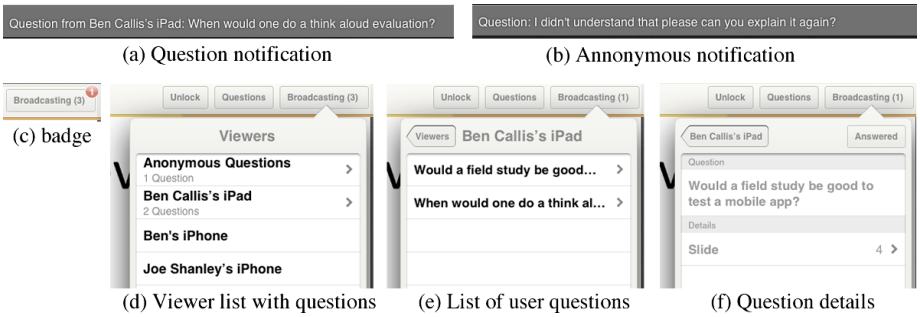


Figure 4.33: Viewer List



(a) Question notification (b) Annonymous notification (c) badge (d) Viewer list with questions (e) List of user questions (f) Question details

Figure 4.34: Views to allow presenter to access viewer's questions

Figure 4.34 demonstrates how the presenter can view user submitted questions. When a presenter receives a question, a subtle notification is temporally displayed at the bottom of their screen. This notification includes the question and the viewer's name if available (figure 4.34a and 4.34b) . The questions are stored alongside the questions author in the viewer list (figure 4.34d). The presenter can view user questions at any time during the presentation by opening the viewers list and navigating to the question detailed screen (figure 4.34e and 4.34f). This screen includes the question title and page number which the presenter can select to view the page. The

screen also includes an ‘Answered’ button which allows the presenter to discard the question.
To inform the presenter of the number of unanswered questions, a notification badge (`MKNumberBadgeView`[37]) is applied to the ‘Broadcasting’ toolbar (figure 4.34c). The notification badge is cleared when the presenter views the questions.

4.3.5 Navigation Lock/Unlock

To allow the presenter to enable\disable viewer navigation an additional packet subclass was created (`IPPacket NavigationLock`). This packet (illustrated in figure 4.35) includes the lock state (`boolean`) and the current page number (`integer`). To support navigation locking the document viewer class was modified to disable gestures for viewers when the presentation is locked. An example of one of the modified methods is shown below.

```
1 - (void)incrementPageNumberAnimated:(BOOL)animated{
2     if ( ! _interactionLocked || ! _isInClientMode ){ // can only change page if navigation is unlocked or a presenter
```

Presenters can toggle the navigation lock the navigation at any time by tapping the ’Lock’\’Unlock’ button located in the document viewer’s toolbar. This action creates an `IPPacketNavigationLock` packet and distributes it reliably to all connected viewers. When the packet has been successfully transmitted the presenter is informed by a custom on screen alert (illustrated in figure 4.36 and 4.37).

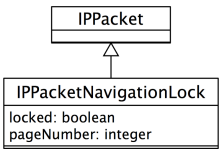


Figure 4.35: Lock packet class

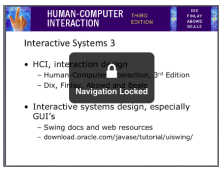


Figure 4.36: Locked alert

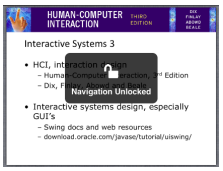


Figure 4.37: Unlocked alert

When a viewer receives a `IPPacketNavigationLock` packet, the `IPBroadcast` class informs the document viewer of the navigation state and notifies the user (shown in figure 4.36 and 4.37). When the navigation is locked, viewers devices are automatically synchronised to the correct page using the `pageNumber` in the packet.

To ensure viewers who join a presentation late have the same locked state, the `IPBroadcast` class was modified to automatically send a navigation packet to viewers when they join. When a presenter ends the broadcast, navigation is automatically unlocked, to allow viewers to explore the content at their own pace.

4.3.6 ‘Open In’ support

To enable the application to open PDF files from other applications on the device, the project’s plist was configured to register support with the system. The modified plist file is included in listing G.4.

Figure 4.38: ’Open in’ support

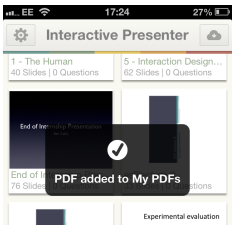


Figure 4.39: PDF successfully added

Figure 4.38 illustrates the application appearing as an option in the ‘open in’ list when a PDF was tapped in the iOS Mail. When the application is selected from this list, it is launched and the file is copied into the

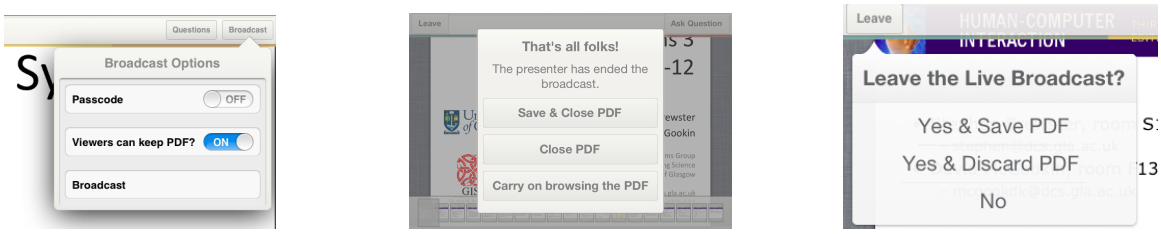
[37] MKNumberBadgeView - `http://www.cocoacontrols.com/controls/mknumberbadgeview`

`inbox` directory, located within the `Documents` folder. During the application launch, the `application:openURL:sourceApplication` method is invoked to inform the application of the path to the file. This method moves the file into the documents directory and adds it into the database. The user is informed once the file has been added via a custom onscreen alert, shown in figure 4.39.

```
1 - (BOOL)application:(UIApplication *) application  openURL:(NSURL *)url sourceApplication:(NSString *)sourceApplication  annotation :( id) annotation {
2     NSString *savedURL = [BCUtility movePDFintoApp:url];
3     if ( savedURL ){ // if  successfully  moved  file  into  documents  directory
4         [_myPDFsViewController addPDFAtPath:savedURL]; // add to  the  grid  view
5         [SVProgressHUD showSuccessWithStatus:@"PDF_added_to_My_PDFs"]; // inform the user of success
6     }else{  [SVProgressHUD showErrorWithStatus:@"Unable_to_add_to_My_PDFs"]; } // inform the user  of  a  problem
```

4.3.7 Store PDF from presentation

To allow presenters to specify if a viewer can save the PDF, an additional field was added to the broadcast options view controller (illustrated in figure 4.40a). Viewers are informed about the presentations saving permission when they join the broadcast. This information is transmitted as a serialised boolean value in an `IPPacketCanSavePDF` packet. The `IPBroadcast` class was modified to store the navigation state. When the presenter or viewer leaves the presentation the navigation state is checked, to determine if the viewer can save the presentation. If the presenter has granted saving permissions, a save option is added to the popover (illustrated in figure 4.40b and 4.40c). When the save option is selected, the PDF file is moved into the *Documents* directory and added to the data model, allowing it to be accessed from the 'My PDFs' view.



(a) Permission set in broadcast options (b) Save PDF when presenter leaves (c) Save PDF when viewer leaves

Figure 4.40: Save PDF from presentation

4.3.8 Live PDF View Modifications

During the initial evaluation users praised the animations used throughout the application. However, one participant suggested that newly discovered broadcasts should animate into the 'Live PDFs' grid view. Based on this feedback, the `IPLivePDFsViewController` was modified to animate the insertion and removal of cells. Figure 4.41 gives a visual representation of the insertion animation. The animation consists of three parts: firstly, the 'No Live PDFs' label is faded out; secondly, a cell is inserted and faded into the view; finally, the notification badge is updated.

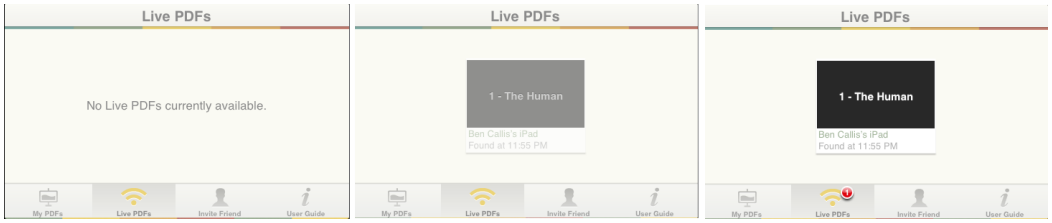


Figure 4.41: Live PDF animation

A short earcon was also added, which sounds when a broadcast is discovered. Blattner defines earcons as "non-verbal audio messages that are used in the computer/user interface to provide information to the user about some computer object, operation or interaction" [5]. The earcon provides auditory feedback which supports the

animation and the badge notification, helping users recognise new broadcasts. The earcon is played at the current system audio volume using the System Sound Services[38] interface. This allows users to control the exact volume of the earcon and disable it as necessary by putting their device in silent mode.

```
1 - (void)matchmakingClient:( IPSearchingPresentationClient *) client serverBecameAvailable :(NSString *)peerID{ // IPLivePDFsViewController
2     if (_noLivePDFsLabel.alpha>0) // if 'No Live PDFS' label is visible remove it gradually over 1/5 th second as there is now a server
3         [UIView animateWithDuration:0.2 animations :^{ _noLivePDFsLabel.alpha = 0.0;} completion :^( BOOL finished){ [_noLivePDFsLabel removeFromSuperview];}];
4     [_gmGridView insertObjectAtIndex :[ _presentationClient availableServerCount ]-1 withAnimation:GMGridViewItemAnimationFade]; // fade cell in
5     dispatch_after ( dispatch_time (DISPATCH_TIME_NOW, 0.3 * NSEC_PER_SEC), dispatch_get_current_queue(), ^{ // tie badge update with animation
6         self . navigationController .tabBarItem.badgeValue = [ self badgeString ]; // update the number badge
7         [ self playBeepSound]; }); // play earcon if device is not on silent
```

4.3.9 User Guide

Figure 4.42: Part of the user guide included within the application

As applications are distributed digitally it is not possible to include a physical user guide. To improve the usability of the application for first time users, a seven page digital user guide was created for the application (four pages are illustrated in figure 4.42). The guide is separated into three sections: importing, presenting and viewing. Each section provides a visual tutorial illustrating how to use the applications key functionality. The complete user guide is included in appendix A. Users can navigate through the guide by scrolling left or right. The guide is implemented using `AFImageViewer`[39], an open source custom view. This view was modified to allow users to change page using the `UIPageControl` situated along the bottom of the view (shown in figure 4.42). The additional view code is shown in listing G.5.

4.4 Summary

Many techniques and technologies have been employed, encountered and learnt about during the implementation of this application. The final product contains 92 classes including 11 external libraries. Information about the key classes and libraries is included in appendix C.
During the implementation phase a large proportion of time was spent developing a highly extensible networking component. This component currently recognises twelve packet types (illustrated in figure 4.43) and allows additional functionality to be implemented by subclassing the `IPPacket` class and modifying a single method in the `IPBroadcast` class.

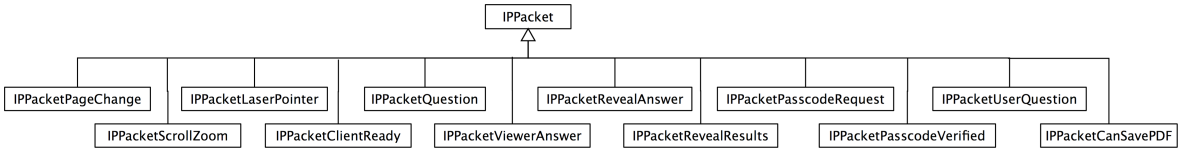


Figure 4.43: 12 packets used in the application

[38] System Sound Services - `http://developer.apple.com/library/ios/#documentation/AudioToolbox/Reference/SystemSoundServicesReference/Reference/reference.html`
[39] AFImageViewer - `https://github.com/AdrianFlorian/AFImageViewer`

Chapter 5

Testing

5.1 Usability Test

Two evaluation approaches were used to evaluate the application after the second implementation cycle: firstly, a short-term usability evaluation which focused on an individual user's experience when interacting with the system; and secondly, a real-world trial in a university lecture that aimed to gather feedback from a lecturer and students when using the application in context.

5.1.1 Short-term Evaluation

The short-term evaluation was designed to test the usability of the applications key features in a short period of time. The evaluation aimed to expose users to both versions of the application whilst testing the presenter and viewer functionally.
Three devices were used during the evaluation to illustrate the networking multiplicity. Each participant was issued an iPad at the start of the evaluation and given the chance to read the in-app user guide to familiarise themselves with the application. Participants were then issued a task sheet containing fourteen short tasks (task sheet is included in appendix H.2). The initial set of tasks required each participant to present a document. During these tasks the evaluator connected an iPhone and an iPad to the broadcast to expose participants to both versions of the application. The second set of tasks required each participant to join a presentation led by the evaluator. During the set of tasks participants were encouraged to describe their thought process ('think aloud') to allow the demonstrator to take notes. Once the participant had completed all tasks they were allowed to explore the application freely on the iPhone and iPad. Participants were then asked to fill in a short questionnaire. This questionnaire included a SUS evaluation and several application specific questions.

The System Usability Scale (SUS) is a simple, ten-item scale giving a global view of subjective assessments of usability. It was developed by John Brooke in 1986 as a tool to be used in usability evaluations of electronic office systems. Brooke describes SUS as "a reliable, low-cost usability scale that can be used for global assessments of systems usability" [6]. Each item in the SUS has has five response options, labeled 1 to 5, which represent *Strongly Disagree* and *Strongly Agree*, respectively. Results from the SUS are combined to yield a single number, which represents a composite measure of the overall usability of the system being studied (full details of how the overall score is calculated is included in H.1). A recent evaluation performed by the Usability Professionals' Association found the SUS to be the most reliable evaluation survey when compared to four other similar questionnaires [14]. The study also showed that an SUS evaluation with a sample size of twelve, would lead to the same conclusion of a bigger sample size.

Results

Twelve participants took part in the short-term evaluation, nine of which were iOS users. The feedback gained through the 'Think Aloud' observations and online questionnaires is summarised below. Additionally, all feedback gained from the online questionnaire is included in appendix H.3.E.

Think Aloud Observations

All participants were able to successfully complete all fourteen tasks. Feedback towards the application was overwhelmingly positive. Many users commented that the application was extremely polished and specifically praised the animations used throughout. A few users had some difficulties adding a question due to a few uninformative button names. For example, one user was unsure if the 'cancel' button would discard the question or dismiss the view. Based on these observations two buttons located on the add question screen had their label changed. This is illustrated in figure 5.1.

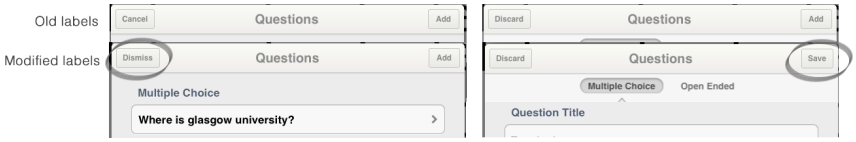


Figure 5.1: Alterations to the question button labels based on feedback.

Two participants found it difficult to locate a file they imported from DropBox. This was mostly due to the user importing the file whilst the 'User Guide' tab was selected. Based on this feedback, the import functionality was modified to automatically present the 'My PDFs' detailed view when a file was imported.

SUS evaluation

The results of the System Usability Scale are shown in the table below. The application scored 89.78 out of a possible 100, over 21 points higher than the average score of 68. This implies the application has 'Excellent' usability, based on the adjective rating scale, defined by the Journal of Usability Studies [4]. The score is illustrated on this scale in figure 5.2.

#	Question	Rating Avg	Score Contribution
1	I think that I would like to use this application frequently.	4.33	3.33
2	I found this application unnecessarily complex.	1.33	3.67
3	I thought this application was easy to use.	4.75	3.75
4	I think that I would need assistance to be able to use this application.	1.50	3.50
5	I found the various functions in this application were well integrated.	4.75	3.75
6	I thought there was too much inconsistency in this application.	1.67	3.33
7	I would imagine that most people would learn to use this application very quickly.	4.75	3.75
8	I found this application very cumbersome/awkward to use.	1.00	4.00
9	I felt very confident using this application.	4.25	3.25
10	I needed to learn a lot of things before I could get going with this application.	1.42	3.58
		SUS Score:	**89.78**

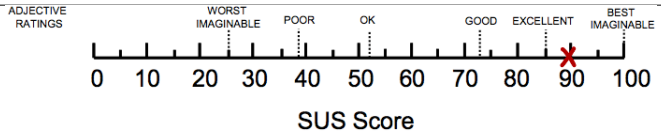


Figure 5.2: The application received an SUS score of 89.78 denoting 'Excellent usability'.

What would you add to Interactive Presenter?

Participants identified eight additional features for the application. Three of the proposed features had previously been identified during the requirements capture.

- ''Ability to annotate slides" (FR 17, 18)
- *"Text based note support"* (FR 13)
- *"File synchronisation across devices"* (FR 16)
- *"Organise saved PDFs into folders"*
- *"Questions without correct answers"*
- *"Point to graphs and highlight answers"*
- *"Audio recording of presentations"*
- *"Linked with something like Google Hangouts*[1] *or similar app so you wouldn't need iOS to join"*

[1]Google Hangouts - `http://www.google.com/+/learnmore/hangouts/`

Multiple participants requested the ability to take notes by annotating the documents directly or using a virtual keyboard. Textual notes support could be implemented using standard Cocoa Touch controls and storing the notes in the core data model. Document annotation is significantly more complex, as custom controls would need to be developed. At the time of writing, no open source libraries were available which offered this functionally. However, it could be implemented by manually tracking gestures and writing them to a PDF file. Unfortunately, this was considered out of scope for the project due to the time constraints. However, note taking and all other features will be considered for future versions of the application.

What did you find difficult when using Interactive Presenter?
Half of the participants stated they had no difficulties using the application. One participant declared that they were confused with the flow of the application at time due to *" a lack of consistency in the names of buttons"*. This issue was directly observed during the 'think aloud' sessions and has since been rectified (illustrated in figure 5.1). Others commented that the user guide was fairly large. However, these users also noted that the user guide was informative and felt that it would only need to be referred to a few times.

When would you imagine using Interactive Presenter?
Participants identified seven environments where they felt the application could be used. Each environment is listed below with the number of times it was identified.

- Lectures - **8**
- Schools (lessons and workshops) - **4**
- Small groups (revision and team meetings) - **4**
- Business meetings - **2**
- Corporate events - **1**
- Marketing and Sales - **1**
- Training days - **1**

As expected, all participants stated the application could be used in a teaching environment. However, participants also identified two areas which had not previously been considered: Marketing and Sales, and Training days. One participant stated the application would be ideal for travelling salesmen as it would allow them to present material to clients without the need to carry hard copies of presentations.

Would you recommend interactive presenter to somebody who wanted to present?
All participants stated they would recommend the application. This statistic is extremely rewarding as it illustrates the success of the application.

5.1.2 University Trial

To assess the application as an educational tool it was used within a lecture at the University of Glasgow and presented to the engineering manager at Liverpool John Moore University (LJMU)[2].

Lecture Trial
The application was tested in a one hour lecture within the Computing Science Department at the University of Glasgow. Three students and one lecturer participated in this trial. The lecture presented the material using an iPad 2. Each student was given a different device with the application installed (iPhone 3gs, iPhone 4 and iPad 2). The goal of the trial was to evaluate the networking performance over a long duration and assess the usability of the application in a real world scenario.

The application worked flawlessly throughout the lecture. Students were able to connect to the broadcast and obtain the PDF within seconds. Synchronisation between devices was fluid throughout the presentation. The lecturer made use of all the presenting functionality within the application and students were able to successfully answer multiple choice questions. Students praised the application and felt the question functionality could add real value to lecturers. After the presentation the lecture made the following comment:

[2]LJMU - `http://www.ljmu.ac.uk`

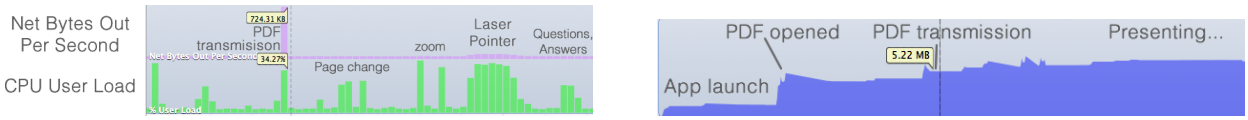
"iPresenter was a very straightforward drop-in replacement for a standard slide show viewer. The UI when viewing PDF's is clean and simple, and it was easy to navigate through the slide deck quickly. The automatic detection of client devices and broadcasting was really neat, and involved no effort on my part whatsoever. Being able to point and broadcast the pointer to all devices worked nicely. Incoming questions from students were discreetly displayed but noticeable. Distributing questions was also easy, although I didn't test this to its full potential, and being able to instantly reflect the results back to the students was useful and easy to do."

University Feedback
The application was demonstrated to Paul Wright an Engineering Manager at LJMU. Feedback was extremely positive:

"After describing this app to some of the staff in the school of engineering, it received very complementary comments and a number of them saw an immediate use for this App. They were interested in using it in tutorial groups and liked the way the master device could send out questions to other devices, allowing them to reflect on their own teaching. They asked about its availability. The app was also described to our schools formula student team (LJMU Racingteam[3]), they saw its potential and asked about its availability for them to use in their three presentations at the 2013 Formula Student competitions in Silverstone, Hockenheim and China. They see that the being able to link devices will allow them to provide more interactive presentations to the judges."

5.2 Performance Testing

During the implementation of the project the application was frequently tested on different devices to judge performance. Although no problems were observed, the application was analysed using the 'Instruments'[4] tool to obtain numerical performance values. The applications CPU usage, outgoing network usage and memory usage was analysed when presenting a PDF on an iPad 2, to four viewers.



(a) Outgoing network and CPU usage

(b) Memory usage

Figure 5.3: Different performance testing metrics

Figure 5.3a illustrates the network usage and CPU usage when presenting a document. The network usage peaked at 724 kilobytes per second (KBps), when transmitting the PDF to a viewer. Once all users had connected the outgoing network usage was minimal, even when navigating around the document. The network usage increased to 4KBps when the laser pointer was used, well within the networks capacity. The number of outgoing bytes per second will increase in correlation to the number of viewers; however, as the usage is minimal, there should be no apparent latency issues when a large number of viewers are connected to a presentation.
CPU usage peaked at 36% when zooming in on the document and using the laser pointer. This CPU usage is likely to be related to updating the display. There was also an apparent CPU spike when splitting the PDF into packets and transmitting it to viewers (34% CPU usage). Throughout the test, the average CPU load was 12%. This low CPU usage implies that the application will perform well on the original iPad, which has an inferior CPU (approximately 65% slower[3]).
Figure 5.3b illustrates the memory usage of the application during the same test. The application used around 2 Megabyte (MB) of memory when displaying the home screen. This increased to just over 5MB when a 2.3MB PDF was opened and transmitted to viewers. During the presentation the memory usage remained consistent at 5MB, and no memory leaks were observed. As all devices capable of running the iOS 5 have at least 256MB of memory, there should be no issues when running the application.

[3]Formula Student is the Europe's biggest educational motorsport event - `http://www.ljmu.ac.uk/racingteam`
[4]Instruments is a performance, analysis, and testing tool for dynamically tracing and profiling iOS code - `http://developer.apple.com/library/ios/#documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/`

Chapter 6

Conclusion

6.1 Summary

The aim of this project was to develop a tool that allows presentations to be conducted in an interactive way using mobile devices, without any additional hardware or configuration. After two cycles of implementation and three usability evaluations, the author is confident that the finished product has met all the high priority requirements identified in Chapter 2. User feedback was extremely positive during the usability tests described in Chapter 5. Several individuals and departments have shown strong interest in the product and are eager for the application to be released. As no defects were identified during the final evaluation, the author plans to release the application into the iOS AppStore in summer 2013.

6.2 Future Development

Once the application has been released, the author will continue to develop the application to incorporate additional features into future releases. Several additional features were identified by potential users during the usability test (Chapter 5, section 5.1), six of which are listed below:

- Annotation support
- File synchronisation across devices
- PDF folder management
- Additional question types
- Audio recording of presentations
- Other platform support

These features were not in the scope of the original project, however; they are likely to be included in future versions of the application.

6.3 Lessons Learnt

Throughout the project the author has become increasingly competent in developing applications for the iOS platform. Creating a universal app which provides the same functionality to all devices proved an exciting challenge, resulting in a deep understanding of iOS application design. During the implementation phase, the author has become a proficient iOS developer with a strong understanding of the Cocoa Touch framework and the Objective-C language. As the application involved a substantial amount of networking code, a solid understanding of the fundamental concepts in networked systems architecture has also been gained.

Bibliography

[1] Idea flight technical approach. http://www.ideaflight.com/2011/06/technical-approach/.

[2] iOS Version Statistics. http://david-smith.org/iosversionstats/. Online; accessed 5/3/2013.

[3] Early ipad 2 apple a5 benchmarks. http://www.iosnoops.com/2011/03/12/early-ipad-2-apple-a5-benchmarks-up-to-65-percent-faster-than-ipad-apple-a4/, 2011. Online; accessed 1/3/2013.

[4] Aaron Bangor, Philip Kortum, James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *JUS — Journal of Usability Studies*, 4:114–123, May 2009.

[5] Sumikawa D. Blattner, M. and R. Greenberg. *Earcons and icons: Their structure and common design principles. Human Computer Interaction.* 1989.

[6] John Brooke. Sus - a quick and dirty usability scale. 1986.

[7] Citrix. Enterprise mobility cloud report. http://citrix.com/content/dam/citrix/en_us/documents/products/q4_enterprise_mobility_cloud_report.pdf, Q4 2012. Online; accessed 15/3/2013.

[8] Joe Conway and Aaron Hillegass. *iOS Programming: The Big Nerd Ranch Guide (Big Nerd Ranch Guides)*. Big Nerd Ranch Guides, 3 edition, 3 2012.

[9] Tim Cook. Apple worldwide developer conference. http://techcrunch.com/2012/10/23/tim-cook-apple-has-sold-100m-ipads-in-just-two-and-a-half-years/, 10 2012. Online; accessed 15/3/2013.

[10] S. Draper, J. Cargill, and Q. Cutts. Electronically enhanced classroom interaction. *Australian journal of educational technology*, 18(1):13–23, 2002.

[11] Darrell Etherington. ios app store boasts 700k apps, 90 http://techcrunch.com/2012/09/12/ios-app-store-boasts-700k-apps-90-downloaded-every-month/. Online; accessed 19/03/2013.

[12] Gartner. Gartner says worldwide mobile phone sales declined 1.7 percent in 2012. http://www.gartner.com/newsroom/id/2335616, 02 2013. Online; accessed 10/3/2013.

[13] Stephen W. Draper and Margaret I. Brown. Use of the prs (personal response system) handsets at glasgow university. http://www.psy.gla.ac.uk/

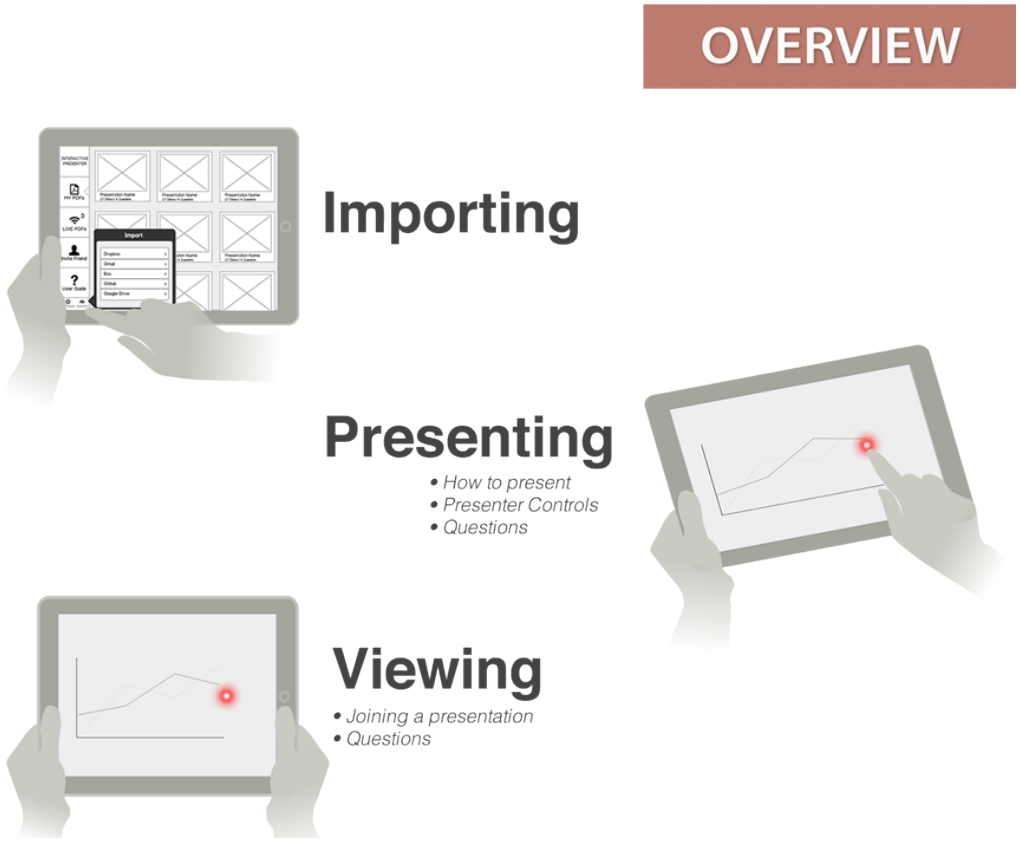
[14] Thomas S. Tullis and Jacqueline N. Stetson. A comparison of questionnaires for assessing website usability. 2004.

[15] Vicki Simpson and Martin Oliver. Electronic voting systems for lectures then and now: A comparison of research and practice. http://ascilite.org.au/ajet/ajet23/simpson.html, 2007. Online; accessed 10/3/2013.

Appendices

Appendix A

User Guide



IMPORT PDFs

Cloud Services

Import from DropBox, GoogleDrive, Box, Gmail or GitHib

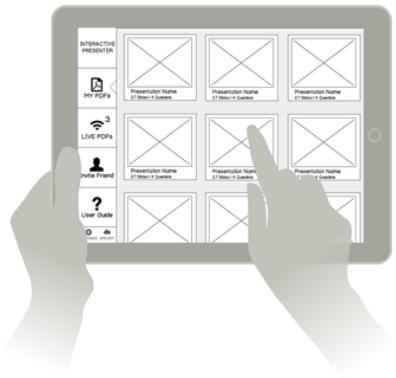
'Open In'

Import from any other Application
such as Mail, Safari or Evernote

iTunes

Import from iTunes File Share
Simply drag the presentation to your device

HOW TO PRESENT



Select PDF

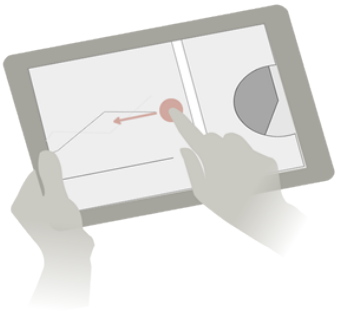
Choose the presentation from 'My PDFs'

Broadcast

Tap 'Broadcast' and set any options...

- *Set a passcode to secure the presentation*
- *Set if viewers can save the presentation*

PRESENTER CONTROLS



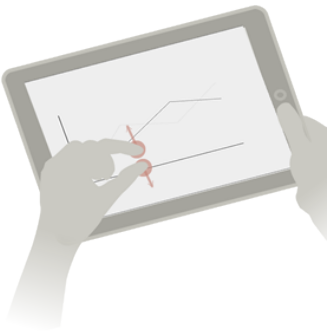
Swipe

to change slide

Or tap on the left or right edge

Pinch

to zoom



Tap & Hold

to use the laser pointer

QUESTIONS

Select

or add a question

Send

To all viewers

See Results

You can reveal just the Answer
or the Answer & the Results

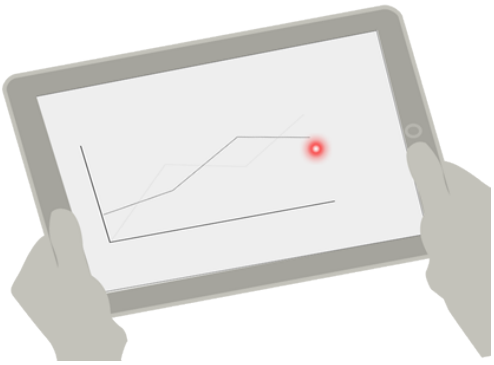
VIEWING

Select PDF

Choose a presentation from 'Live PDFs'
Input the passcode if required

Enjoy

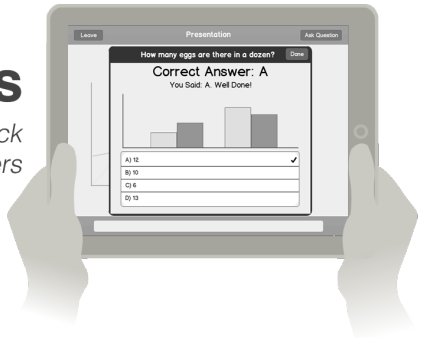
Sit back & enjoy the presentation



QUESTIONS

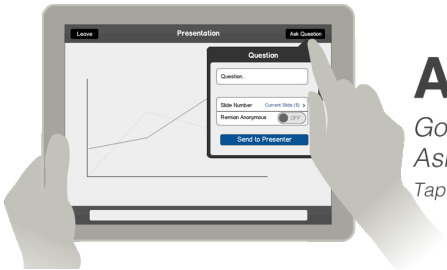
Answer

Presenters can ask you questions throughout the presentation
Select & submit an answer



Results

Get instance feedback on your answers



Ask

Got a question? Ask the presenter
Tap the 'Question' button

Appendix B

iOS

Introduction

iOS is a mobile operating systems developed and distributed by Apple [1]. iOS was previously known as iPhone OS when it was originally released in 2007 for the iPhone and iPod Touch under. Since it's launch it has been extended to support two other Apple devices: the iPad and Apple TV. Unlike other major mobile operating systems, Apple does not licence out iOS to third party manufactures. This allows the iOS operating system to be tightly integrated with the hardware to improve performance. Initially it was not possible for developers to write native application for iOS. Apple released an official software development kit (SDK) alongside iOS 2.0, allowing the development of third party applications. Applications can only be officially distributed through the Apple AppStore. As of September 12, 2012, the store contained over 700,000 applications, making it the largest mobile ecosystem. Developers can currently build applications for two different form factors: mobile (iPhone and iPod touch) and tablet (iPad).

Figure B.1: Devices capable of running iOS 5

As Apple has total control over the hardware and software, iOS updates can be released to devices in a timely manor. Apple tends to provide each device at least two major software updates, resulting in an ecosystem with minimal fragmentation. This streamlines the application development process, as developers need not worry about supporting hundreds of different devices running different versions of the operating system - a common problem in other mobile operating systems such as Android.

There have been six major releases of the operating system, with the latest named iOS 6. Each major release includes a significant number of new user features and hundreds of new APIs. With the released of iOS 5 Apple

[1] Apple Inc. - `http://www.apple.com/uk/`

improved the LLVM [2] compiler, adding support for automatic reference counting (ARC). ARC combines the simplicity of garbage collection with the efficiency of manual memory management. The compiler achieves this by automatically inserting the necessary memory management calls at compile time. Figure B.1 shows all the devices which are capable of running iOS 5. Due to the limited screen real-estate on many of these devices, animation is used frequently to give context to actions. The SDK exposes multiple APIs, allowing developers to use animations in their applications.

Development

This section gives an insight into iOS development and explains the most common iOS design patterns.

Objective-C

iOS applications are written in the Objective-C language using the Cocoa Touch library. Objective-C is an object orientated extension to the C language. All C syntax and functions can be used in Objective-C files. Coca Touch is a UI framework created by Apple. It acts as layer of abstraction to lower levels of the system, providing 3 main features: Animation, Multitasking and Gesture recognisers.

Development Tools

The iOS SDK comes bundled with a suite of software developments tools for OS X (Apple's desktop OS) developed by Apple. These tools are used to develop software for iOS and OSX . The main application in this suite is Xcode - an integrated development environment (IDE). This tool helps developers write/debug code and create user interfaces with interface builder (built into Xcode). Applications can be deployed to devices for testing or the built in iOS simulator can be used.
Instruments is also included in the suite. Instruments is a performance analyser and visualise. This tool make it possible to monitor CPU usage and Memory allocations in real-time.

(a) Xcode IDE (b) iOS Simulator (c) Instruments analysing tool

Figure B.2: The Xcode development tools package

Design Patterns

Model-View-Controller

The Model-View-Controller (MVC) design pattern assigns objects in an application one of three roles: *model*, *view*, or *controller*. This enhances the reusability of objects as views are not directly coupled to specific data models. Figure B.3a gives a visual representation of the MVC.

[2] LLVM is a compiler infrastructure written in C++; it is designed for compile-time, link-time, run-time, and "idle-time" optimisation of programs written in arbitrary programming languages.

Model objects hold data specific to the application and often contain methods that can manipulate and process the data. Model objects do not directly communicate with view objects, instead its data is made available to controller objects.
View objects present information stored in model objects to the user. This information is not accessed directly. Instead controller objects are used as a mediator, instructing view objects of changes in model objects. When users interact with view objects a message is sent to the controller to notify it of any updates.
Controller objects manage the communication between view objects and model objects. When users interaction with view objects, controller objects interpret these actions and communicate new or changed data to the relevant model objects.

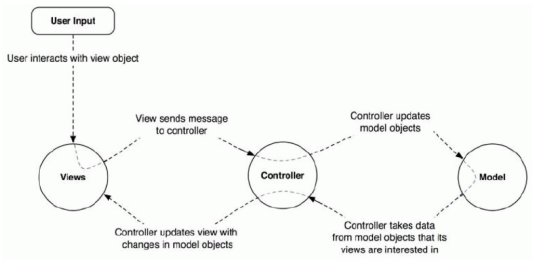
Delegation
Delegation is a simple and powerful pattern in which one object hands over a task to another object. The delegating objects maintains a reference to the other object (known as the delegate) and send messages to it whenever appropriate. The delegate may respond to a message call by updating it's state or appearance and can return a value to it's caller.

An example of an object which makes use of delegation is the UITableView class (part of the Cocoa Touch framework) is illustrated in figure B.3b. A TableView frequently sends messages to its delegate to inform it of events or to request information. Figure B.3b illustrates two of these messages.

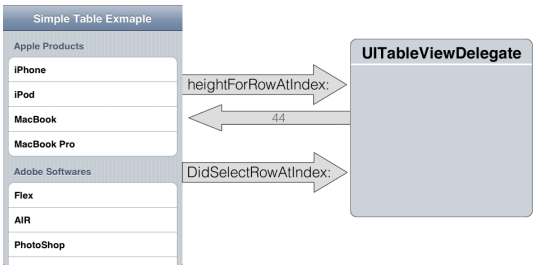
HeightForRowAtIndex is an example of a message where a return value is sent back to it's caller. It allows the delegate to specify the height of a given cell in the table

DidSelectRowAtIndex is an example of a method call with no return value. The delegate often responds to this message by updating the current view or pushing a new view onto the screen.

The main value of delegation is that it allows an object to be customised by another object, without editing the original objects code and without subclassing it. It is also possible for a single object to be a delegate to multiple other objects.



(a) Modal-View-Controller design pattern [8]



(b) TableView delegation example

Figure B.3: Common iOS design patterns

Appendix C

Application files

This appendix describes the Libraries used and some of the key classes within the application.

C.1 Libraries Used

Library	**Purpose**	**License**
GMGridView	Grid view which supports iOS 4 +.	MIT Open Source
VFR	PDF Document Viewer	MIT Open Source
BlockAlert	Customisable Alert View	MIT Open Source
JSNotifier hline SVProgressHUD	Simple notifications Lightweight progress HUD	Apache 2.0. MIT Open Source
FilePicker	Online cloud service integration	MIT Open Source
AFImageViewer	iOS image viewer using UIScrollView and UIPageControll	MIT Open Source
ACFileTransfer	Class to split up files into packets.	Apache 2.0
GrowingTextView	UITextView which grows/shrinks with the text	MIT Open Source
MKNumberBadge	A view that replicates and extends the number badge UI element in iOS	Apache 2.0.
SDSegmentedControl	A drop-in replacement for UISegmentedControl that mimic iOS 6 AppStore tab controls	MIT Open Source

C.2 Main View Controllers

View Controller	Description
IPCustomNavigationVC	Manages iPad view hierarchy and provides custom navigation bar.
BCArrowTabBarVC	Customised TabBar controller which includes an animated tab selected indictor. Used on iPhone.
IPMyFilesVC	Displays documents in a grid to users. Accessible from 'My PDFs' tab.
IPLivePDFsVC	Displays live presentations to user. Controls UI to connect to a presentation.
IPInviteFriendVC	iPhone view controller to allow users to invite friends via SMS or Email.
IPInviteFriendGridVC	iPad view controller to allow users to invite friends.
IPUserGuideVC	Controls the display and interaction with the User Guide.
IPPDFVC	Displays PDF and controls all user interaction (page change, zoom, pan and laser). Communicates with the network to communicate with other devices. Presents many other views during a presentation. Allows interaction to be locked as needed.
IPBroadcastSetupVC	Allows users to setup and start a broadcast. Allows passcode to be set.
IPQuestionListVC	Displays list of questions attached to a presentation. Allows presenters to add or select a question.
IPVAddViewQuestionVC	Manages adding or displaying of questions attached to a presentation. Allows presenters to add or send a question.
IPQuestionSendingVC	Displays results to a question sent out to the audience. Allows answer or results to be sent to audience.
IPQuestionRecivedVC	Displays on a viewers device when they receive a question. Allows an answer to be submitted.
IPResultsVC	Displays answer and results for a question to a viewer. Presented when the presenter reveals the results.
IPAnswerVC	Displays answer for a question to a viewer. Presented when the presenter reveals the answer.
IPUserRaisedQuestionVC	Allows viewers to send questions to the presenter. Users can input a question, select a slide number and state if they would like to remain anonymous.
IPSlideSelectTableVC	Manages a list of slides allowing a viewer to select a slide when sending a question.
IPBroadcastViewersVC	Allows presenter to see a list of viewers. Provides access to any questions raised by viewers.
IPViewerQuestionListVC	Manages a list of questions from a single viewer.
IPViewerSpecificQuestionDetailsVC	Displays details of a single question from a viewer. Allows presenter to navigate to navigate to the question page number, and allows presenter to mark it as answered.

C.3 Networking Classes

Class Name	Description
IPSearchingPresentationClient	Automatically searches for active presentations when the app is launched. Allows connections to be established to a presenter. Re-assembles the PDF received from the presenter.
IPBroadcast	Presenters a presenter or viewer in the live presentation broadcast. Provides all the methods to interact between devices. Communicates with PDF VC to receive actions and to update the UI.
IPViewer	Represents a viewer. Includes peer ID, display name and list of questions.
IPPacket	Base type for all packets. Acts a factory to generate concrete packets. Includes methods to append the header to all packets and to convert a packet of raw data back into a concrete packet.
IPPackerPageChange	A packet sent from a presenter to viewers containing the page number. Generated when presenter changes page.
IPPacketScrollZoom	A packet sent from a presenter to viewers containing the visible rectangle of document. Generated when presenter zooms or pans around a page.
IPPacketLaserPointer	A packet sent from a presenter to viewers containing the position of laser pointer. The point is relative to the document so it will display in the same place on all devices.
IPPacketClientReady	A packet sent from a viewer to the presenter to indicate they are ready to receive action packets. When the presenter receives this packet it automatically sends the current state to ensure everything is synchronised.
IPPacketQuestion	A packet sent from a presenter to viewers containing a question and possible answers. Generated when the presenter sends out a question.
IPPacketViewerAnswer	A packet sent from a viewer to a presenter containing the questions and answer. Generated when a viewer submits an answer to a questions.
IPPacketRevealAnswer	A packet sent from a presenter to viewers containing a questions and correct answer. Generated when a presenter reveals an answer to a question.
IPPacketRevealResults	A packet sent from a presenter to viewers containing a question, possible answers, correct answer and audiences results. Generated when a presenters shares the results to a question.
IPPacketPasscodeRequest	A packet sent from a presenter containing an encrypted passcode. This packet is generated when a presentation is passcode secured and a viewer tries to join.
IPPacketPasscodeVerified	A packet sent from a viewer to presenter to indicate that the passcode has been successfully verified. This starts the PDF transfer process.
IPPacketUserQuestion	A packet sent from a viewer to the presenter when they have a question. The packet includes the question, page number and an anonymous boolean value.
IPPacketNavigationLock	A packet sent from a presenter to viewers when the navigation lock state is changed. The packet includes a state boolean value and the page number so viewers can return the the presenters page when the navigation is locked.
IPPacketCanSavePDF	A packet sent to viewers from the presenter. This packet indicates if a viewer can save the PDF at the end of the presentation.

C.4 Miscellaneous

Class Name	Description
BCBarChart	Custom UIView which draws an elegant colourful bar chart. Used to visualise audiences responses to questions. Can be used in any other project.
BCUtility	Includes several static utility methods for managing files within the sandbox, manipulating images, escaping strings and encrypting strings.
TPConstants	Includes common macros and application constants such as frames, strings and colours. This file is automatically imported into all classes in the application.

C.5 Visual representation of key View Controllers

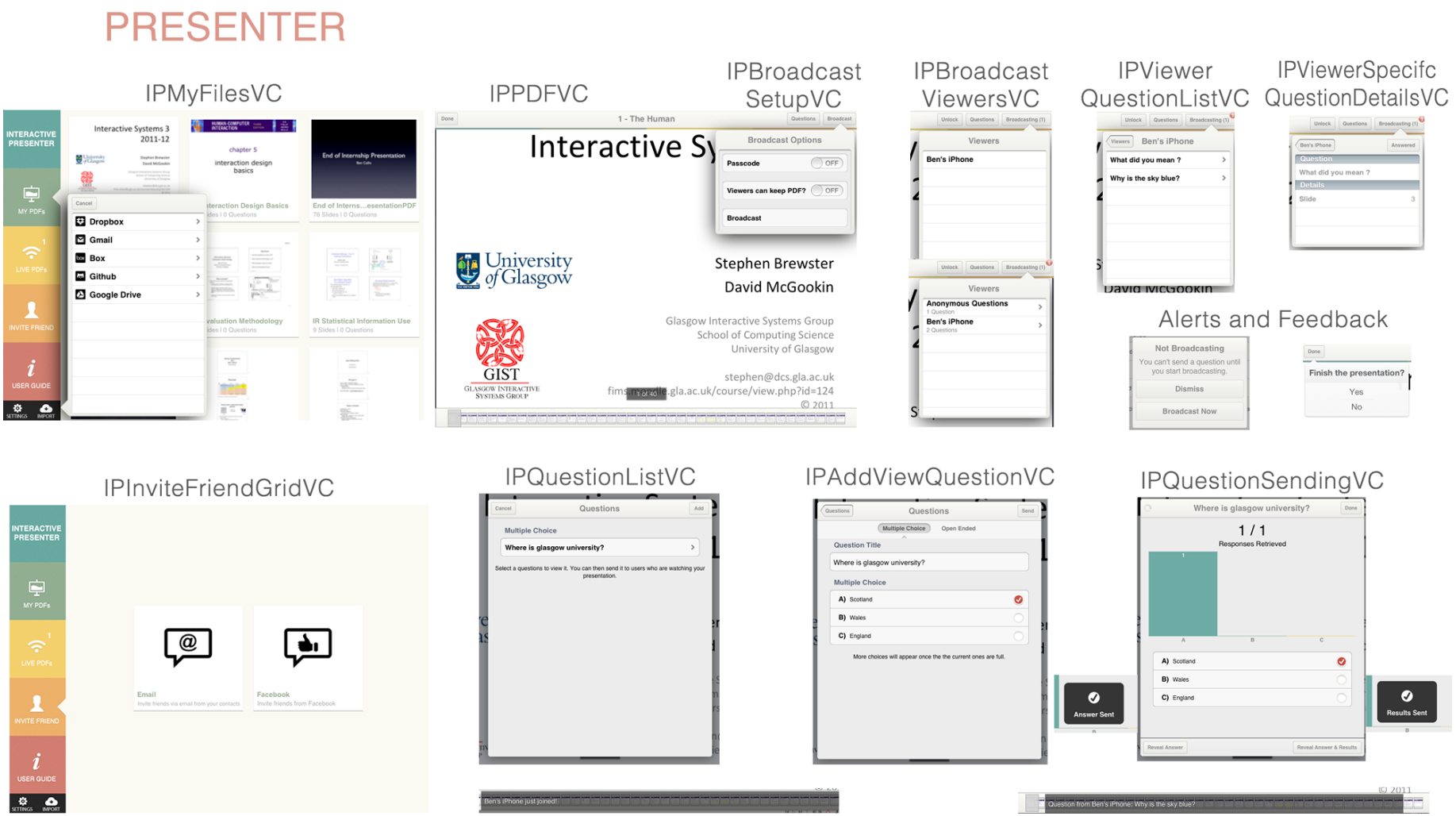


Figure C.1: Major views accessible when presenting

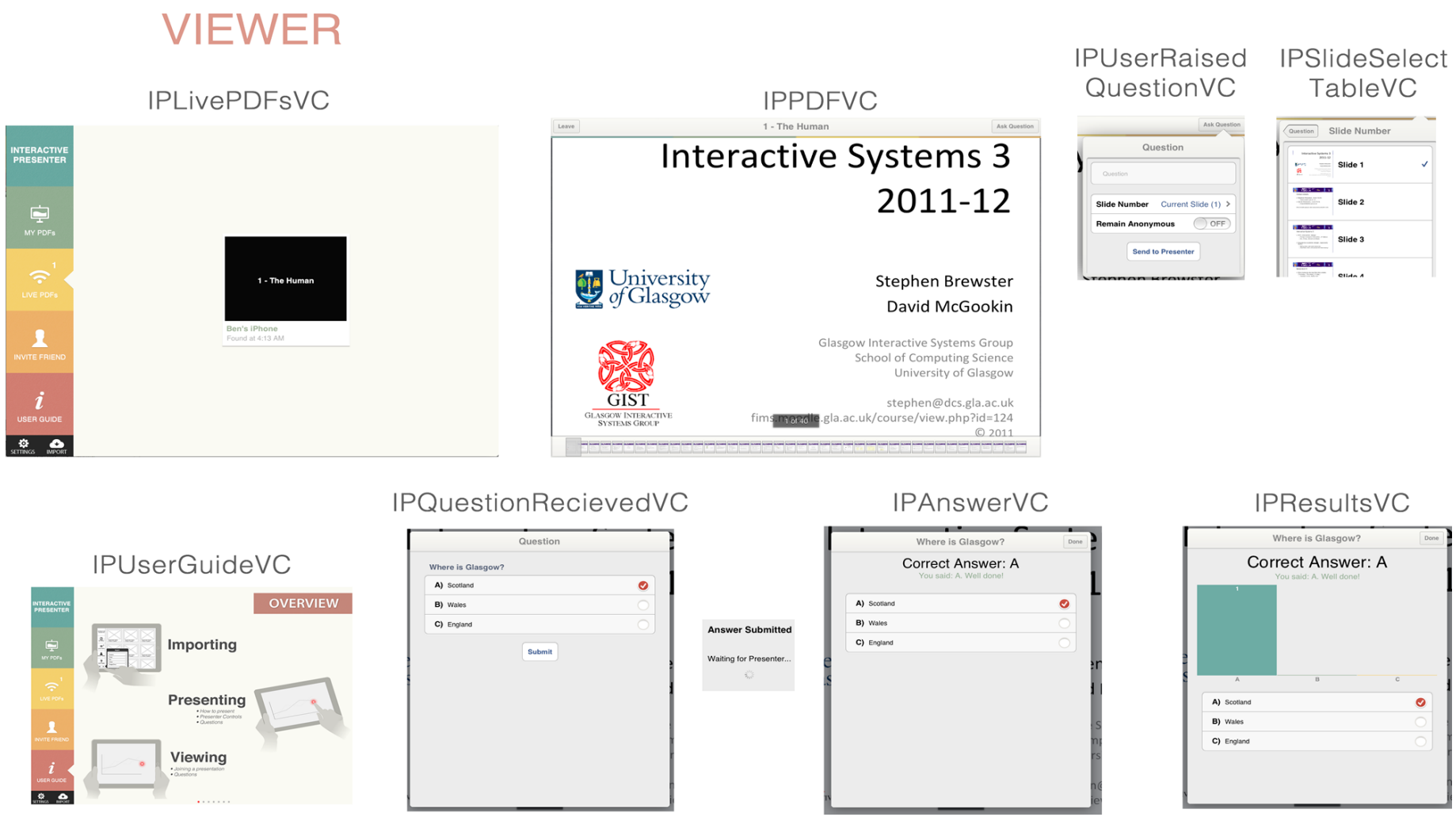
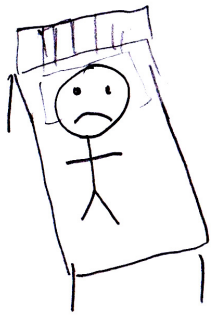


Figure C.2: Major views accessible when viewing a presentation

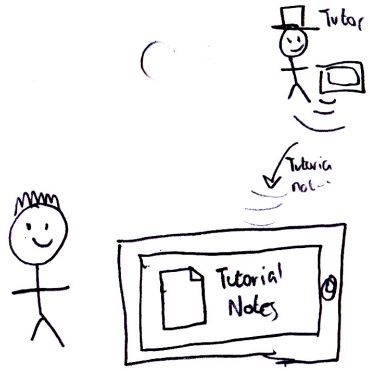
Appendix D

Full Size Storyboards

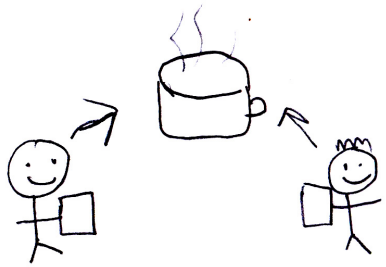
This appendix includes three large storyboards to accompany section 3.1.5 on page 11.



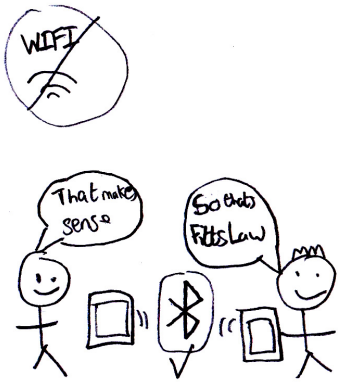
Jim is sick and misses a tutorial.



His friend Ben attends the tutorial. He has the notes on his iPad.

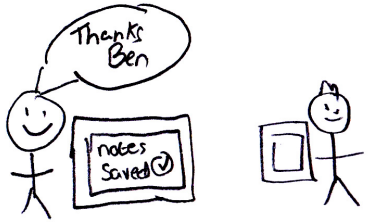


Jim gets better and arranges to meet Ben in a coffee shop. They both bring their iPads.



Ben broadcasts the notes to Jim and talks him through them. This is done seamlessly over bluetooth as the wifi in the shop is broken.

Henry trys to access the Broadcast without asking Ben or Jim. Thankfully he can't access the notes as Ben added a passcode to the broadcast



Jim now has the notes saved on his iPad and has a firm understanding of their contents.

Figure D.1: Personal use scenario

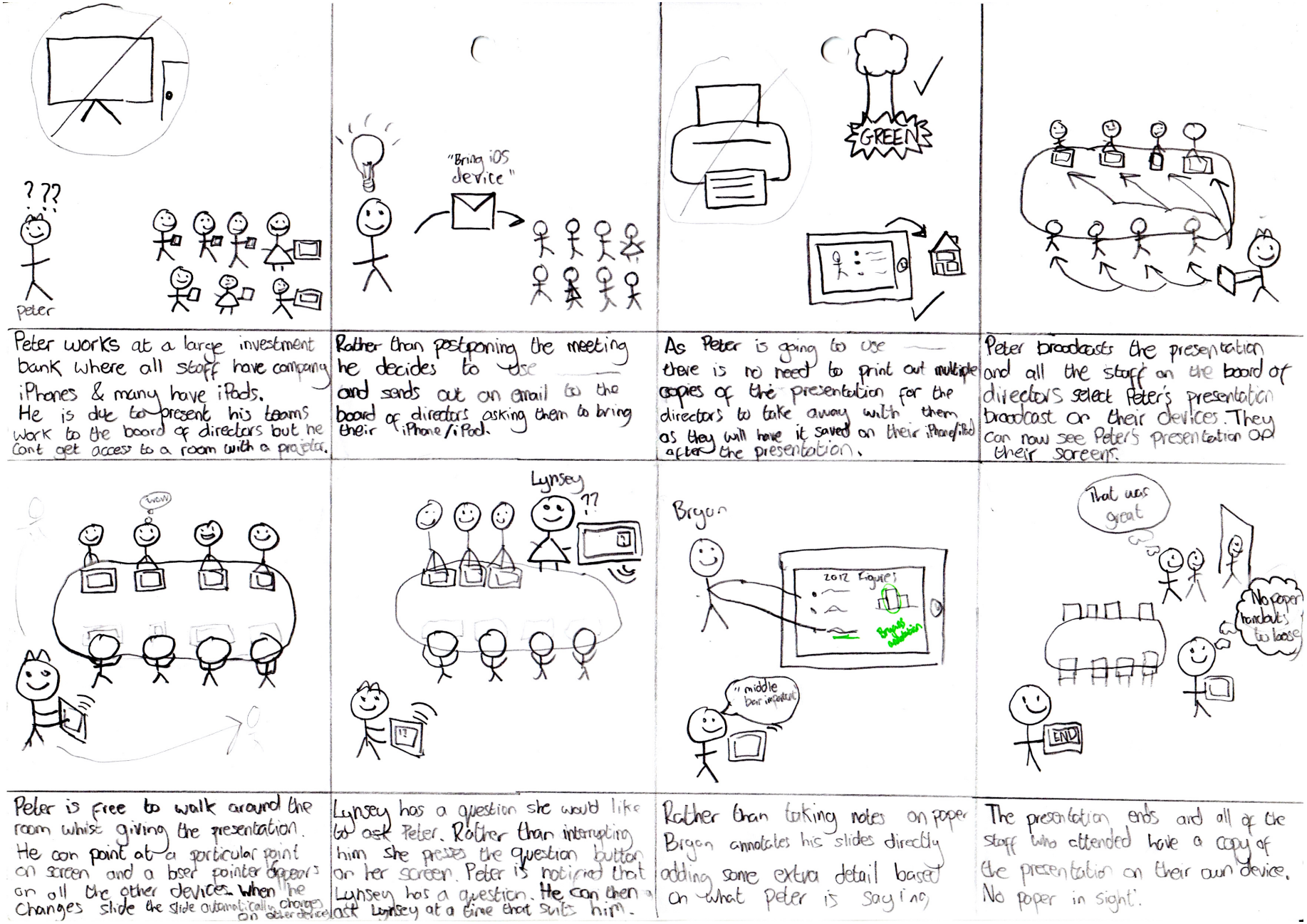


Figure D.2: Academic use scenario

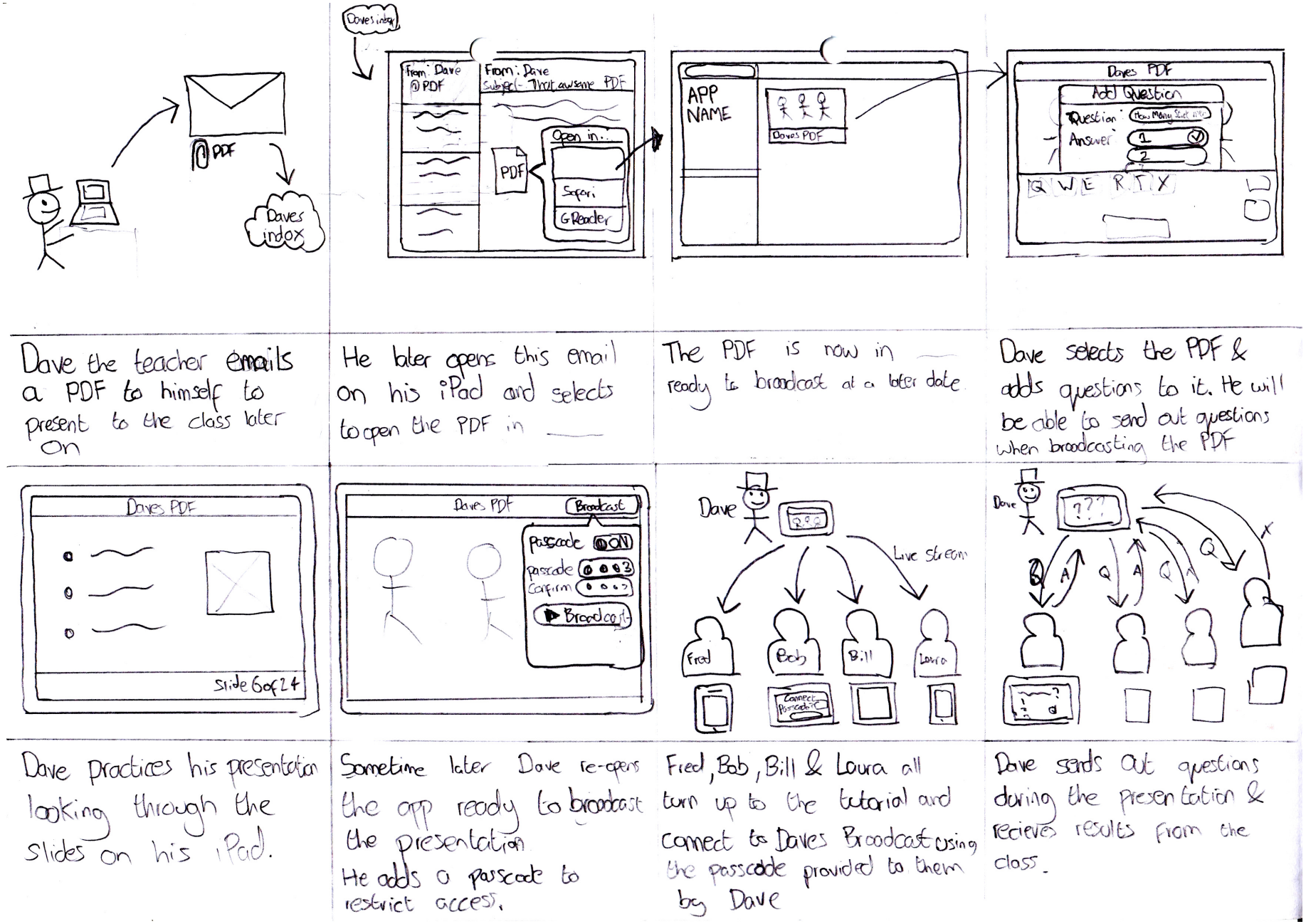


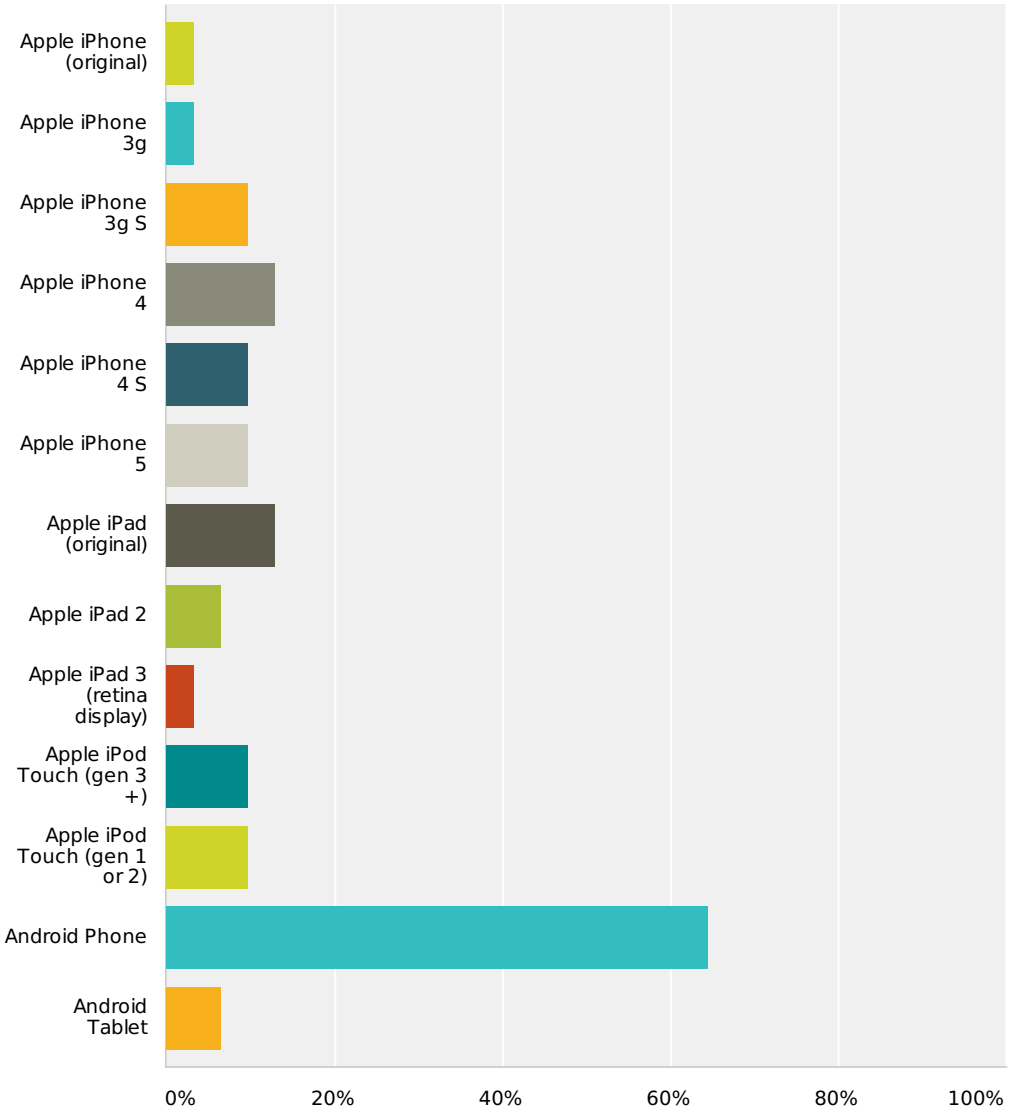
Figure D.3: Business use scenario

Appendix E

Requirements Survey

Q1 Which of the following devices do you own? (Can choose multiple)

Answered: 31 Skipped: 3

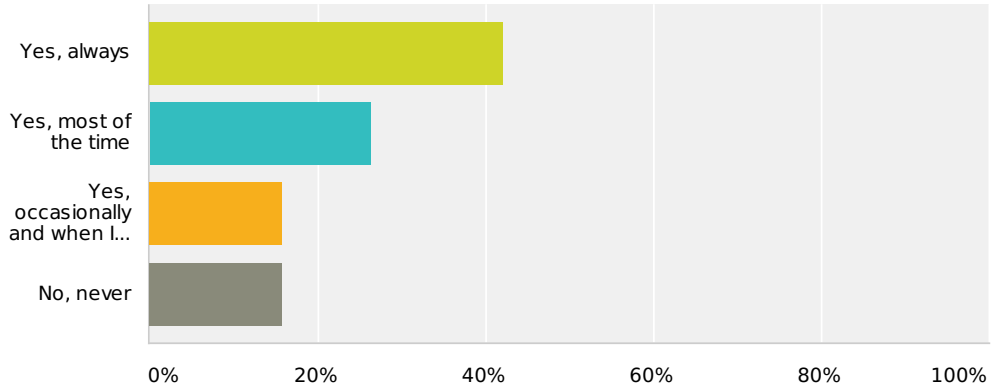


Answer Choices	Responses	
Apple iPhone (original)	**3.23%**	1
Apple iPhone 3g	**3.23%**	1
Apple iPhone 3g S	**9.68%**	3
Apple iPhone 4	**12.90%**	4
Apple iPhone 4 S	**9.68%**	3
Apple iPhone 5	**9.68%**	3
Apple iPad (original)	**12.90%**	4
Apple iPad 2	**6.45%**	2
Apple iPad 3 (retina display)	**3.23%**	1
Apple iPod Touch (gen 3 +)	**9.68%**	3
Apple iPod Touch (gen 1 or 2)	**9.68%**	3
Android Phone	**64.52%**	20
Android Tablet	**6.45%**	2

Total Respondents: 31

Q2 If you have an Apple iOS device, do you frequently carry this with you when you leave the house? (if you have more than one iOS device, do you carry at least one of them with you)

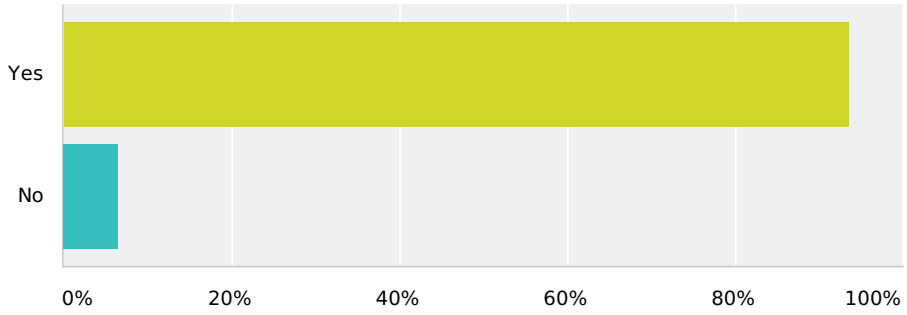
Answered: 19 Skipped: 15



Answer Choices	Responses	
Yes, always	**42.11%**	8
Yes, most of the time	**26.32%**	5
Yes, occasionally and when I know I need it	**15.79%**	3
No, never	**15.79%**	3
Total		19

Q3 Do you download apps for your device?

Answered: 31 Skipped: 3

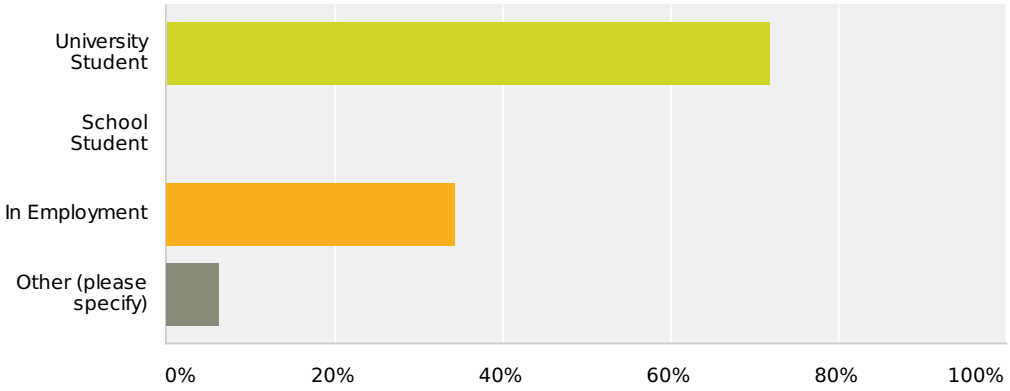


Answer Choices	Responses	
Yes	**93.55%**	29
No	**6.45%**	2
Total		31
Other (please specify) (0)		

#	Other (please specify)	Date
	There are no responses.	

Q4 Which of the following categories do you fall into?

Answered: 32 Skipped: 2

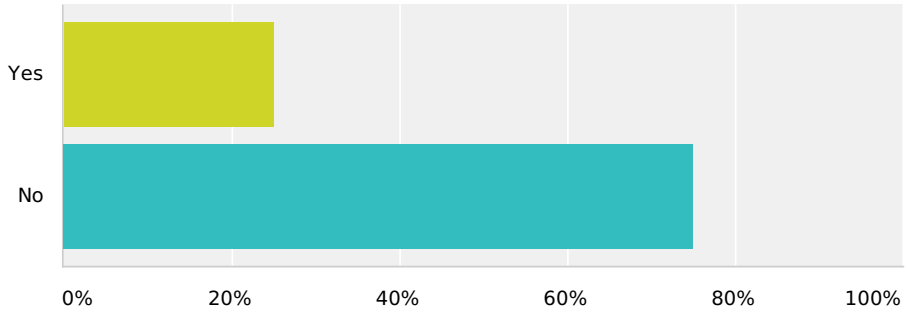


Answer Choices	Responses	
University Student	**71.88%**	23
School Student	**0%**	0
In Employment	**34.38%**	11
Other (please specify)	**6.25%**	2
Total Respondents: 32		

#	Other (please specify)	Date
1	Disabled, unable to work	10/31/2012 3:45 PM
2	unemployed graduate	10/30/2012 3:51 PM

Q5 Have you ever used interactive voting handsets when presenting?

Answered: 32 Skipped: 2



Answer Choices	Responses	
Yes	**25%**	8
No	**75%**	24
Total		32

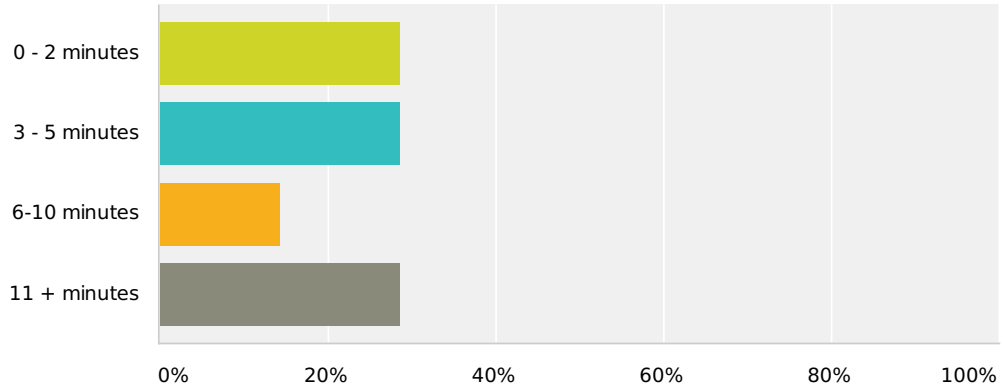
Q6 Which Interactive Voting Systems have you used to present?

Answered: 4 Skipped: 30

#	Responses	Date
1	Active inspire voting pods.	11/29/2012 9:46 AM
2	Activ Inspire	11/20/2012 6:39 AM
3	The CS1P one	10/30/2012 3:21 PM
4	Smarttech Senteo	10/30/2012 11:10 AM

Q7 How long did the hardware take to setup before the presentation?

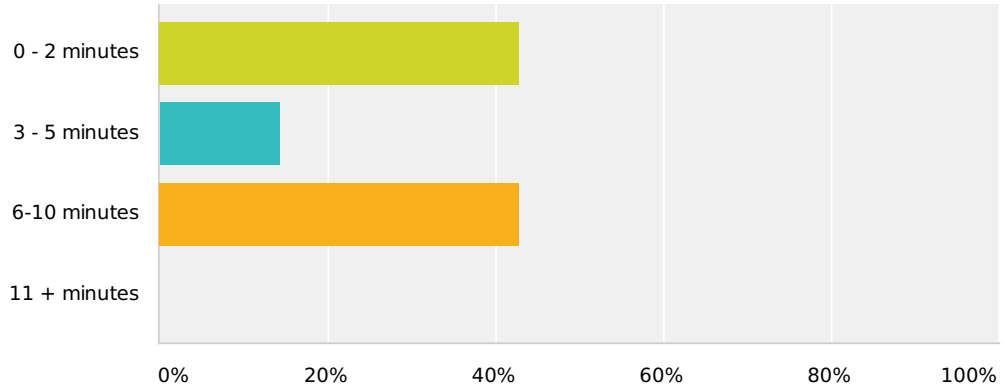
Answered: 7 Skipped: 27



Answer Choices	Responses	
0 - 2 minutes	**28.57%**	2
3 - 5 minutes	**28.57%**	2
6-10 minutes	**14.29%**	1
11 + minutes	**28.57%**	2
Total		7

Q8 How long approximately did it take add the first question to your presentation (including the time it took to link the voting software to your presentation)?

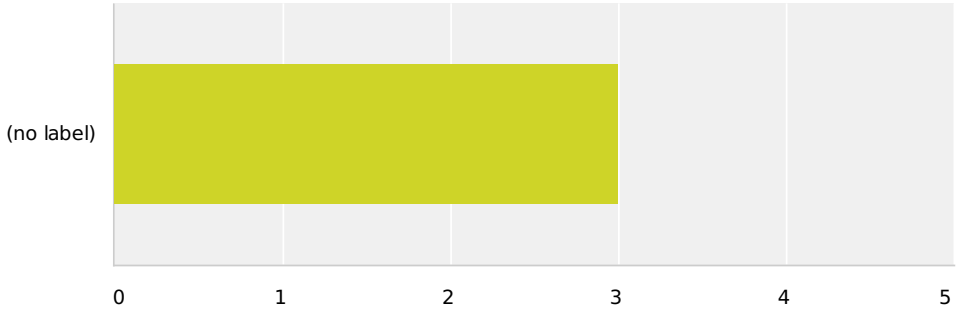
Answered: 7 Skipped: 27



Answer Choices	Responses	
0 - 2 minutes	**42.86%**	3
3 - 5 minutes	**14.29%**	1
6-10 minutes	**42.86%**	3
11 + minutes	**0%**	0
Total		7

Q9 Do you feel using the interactive voting system helped your audience become more engaged in the presentation/meeting/lecture?

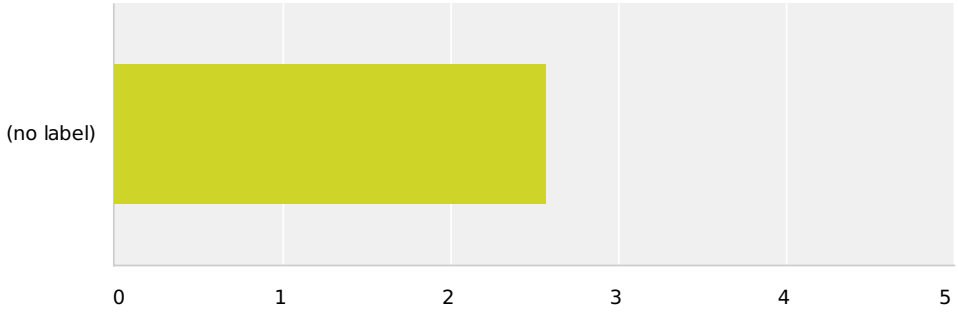
Answered: 7 Skipped: 27



	They appeared less engaged	It appeared to have no positive or negative effect	They appeared somewhat more engaged	They appeared much more enaged	Total	Average Rating
(no label)	**0%** 0	**28.57%** 2	**42.86%** 3	**28.57%** 2	7	3.00

Q10 Do you feel using the interactive handsets helped your audience in understanding the given topic?

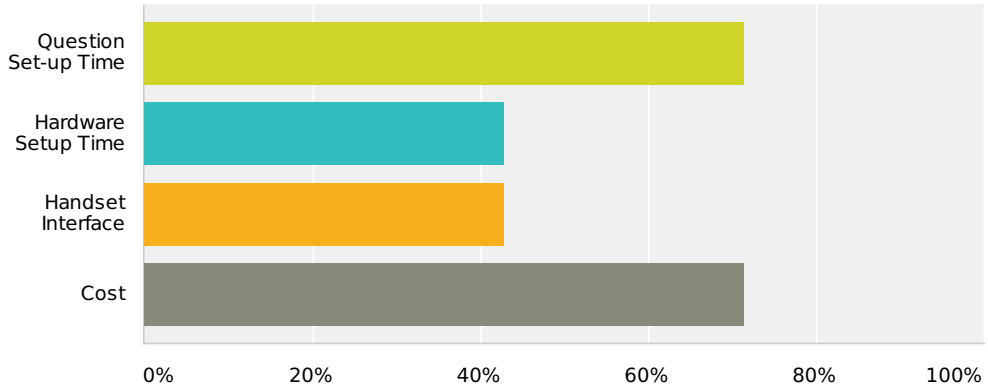
Answered: 7 Skipped: 27



	It appeared to make the topic harder to understand	It had no positive or negative effect	It appeared to help them somewhat in understanding the topic	It appeared to help them greatly in understanding the topic	Total	Average Rating
(no label)	**0%** 0	**57.14%** 4	**28.57%** 2	**14.29%** 1	7	2.57

Q11 What do you think could be improved? (Can choose multiple)

Answered: 7 Skipped: 27

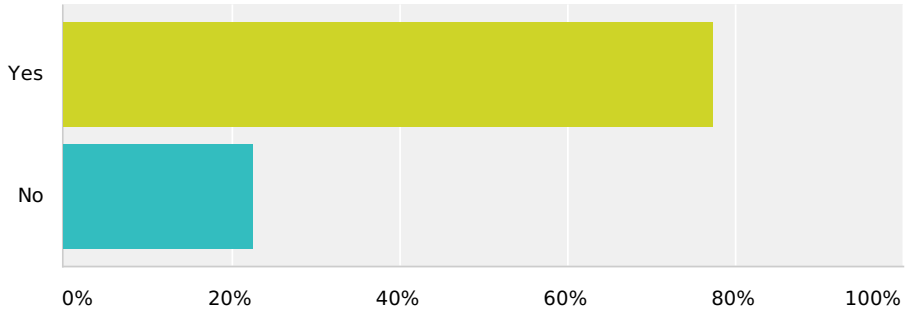


Answer Choices	Responses	
Question Set-up Time	**71.43%**	5
Hardware Setup Time	**42.86%**	3
Handset Interface	**42.86%**	3
Cost	**71.43%**	5
Total Respondents: 7		
Other (please specify) (1)		

#	Other (please specify)	Date
1	I feel that the handsets are sometimes seen as a toy or a gimick. They do have real value but it can be lost on some pupils who just want a "game" instead of a lesson.	11/29/2012 9:46 AM

Q12 Have you ever used interactive voting handsets when VIEWING a presentation or in a meeting?

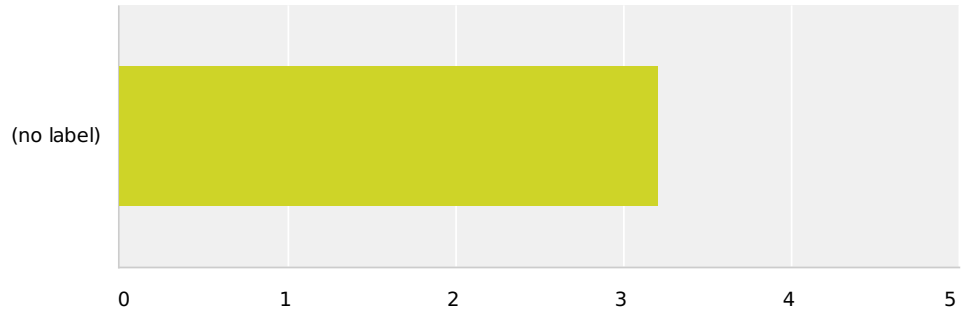
Answered: 31 Skipped: 3



Answer Choices	Responses	
Yes	**77.42%**	24
No	**22.58%**	7
Total		31

Q13 Do you feel using the interactive handset helped you to become more engaged in the presentation/meeting/lecture?

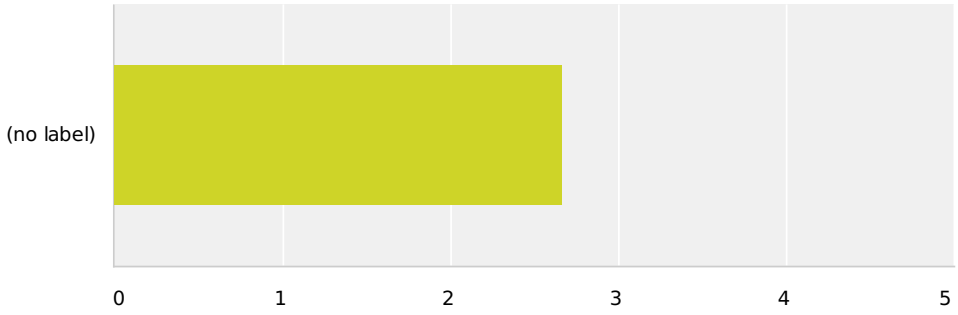
Answered: 24 Skipped: 10



	I felt less engaged	It had no positive or negative effect	I felt somewhat more engaged	I felt much more engaged	Total	Average Rating
(no label)	**4.17%** 1	**4.17%** 1	**58.33%** 14	**33.33%** 8	24	3.21

Q14 Do you feel using the interactive handset helped you to understand the given topic?

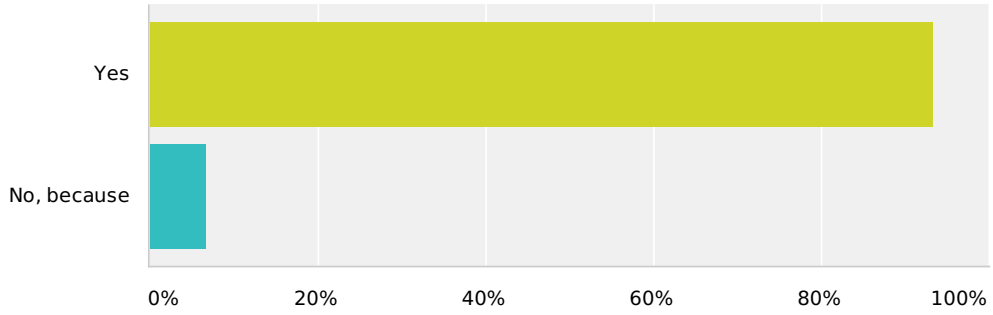
Answered: 24 Skipped: 10



	It made the topic harder to understand.	It has no positive or negative effect	It helped me somewhat in understanding the topic	It helped me greatly in understanding the topic	Total	Average Rating
(no label)	**4.17%** 1	**45.83%** 11	**29.17%** 7	**20.83%** 5	24	2.67

Q15 Would you be open to using a voting system as a VIEWER in a presentation/meeting?

Answered: 30 Skipped: 4

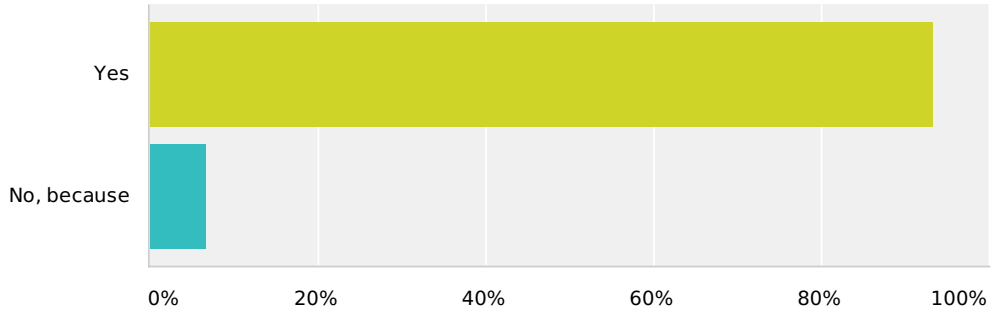


Answer Choices	Responses	
Yes	**93.33%**	28
No, because	**6.67%**	2
Total		30

#	No, because	Date
1	I honestly don't see how this can be better than talking to people.	11/21/2012 6:17 AM
2	It's just a gimmick.	10/30/2012 3:33 PM

Q16 Would you be open to using a voting system when LEADING a presentation/meeting?

Answered: 30 Skipped: 4



Answer Choices	Responses	
Yes	**93.33%**	28
No, because	**6.67%**	2
Total		30

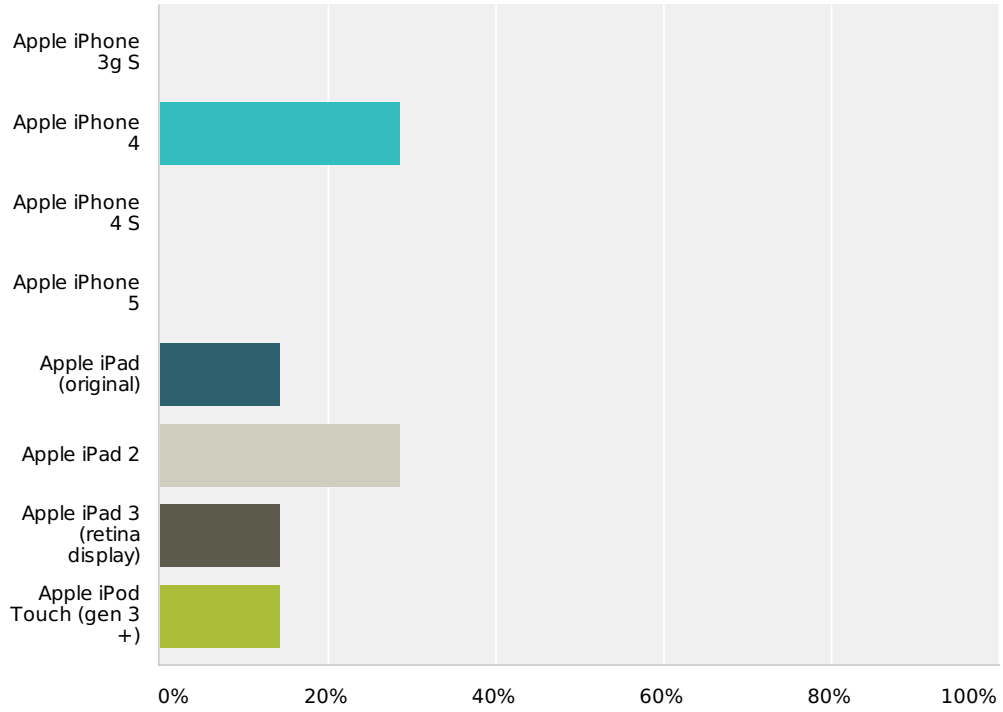
#	No, because	Date
1	See above	11/21/2012 6:17 AM
2	It adds nothing of value, and wastes a lot of time.	10/30/2012 3:33 PM

Appendix F

User Feedback and Refinement Survey

Q1 Which device did you use to test the application?

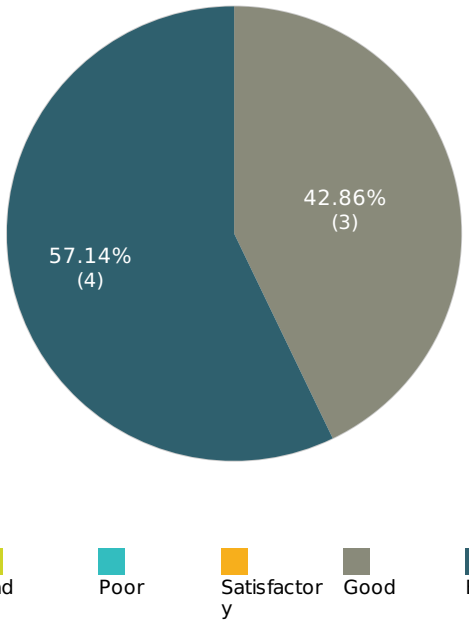
Answered: 7 Skipped: 0



Answer Choices	Responses	
Apple iPhone 3g S	**0%**	0
Apple iPhone 4	**28.57%**	2
Apple iPhone 4 S	**0%**	0
Apple iPhone 5	**0%**	0
Apple iPad (original)	**14.29%**	1
Apple iPad 2	**28.57%**	2
Apple iPad 3 (retina display)	**14.29%**	1
Apple iPod Touch (gen 3 +)	**14.29%**	1
Total		7

Q2 What did you think of the user interface of the application?

Answered: 7 Skipped: 0



	Bad	Poor	Satisfactory	Good	Excellent	Total	Average Rating
(no label)	**0%** 0	**0%** 0	**0%** 0	**42.86%** 3	**57.14%** 4	7	4.57

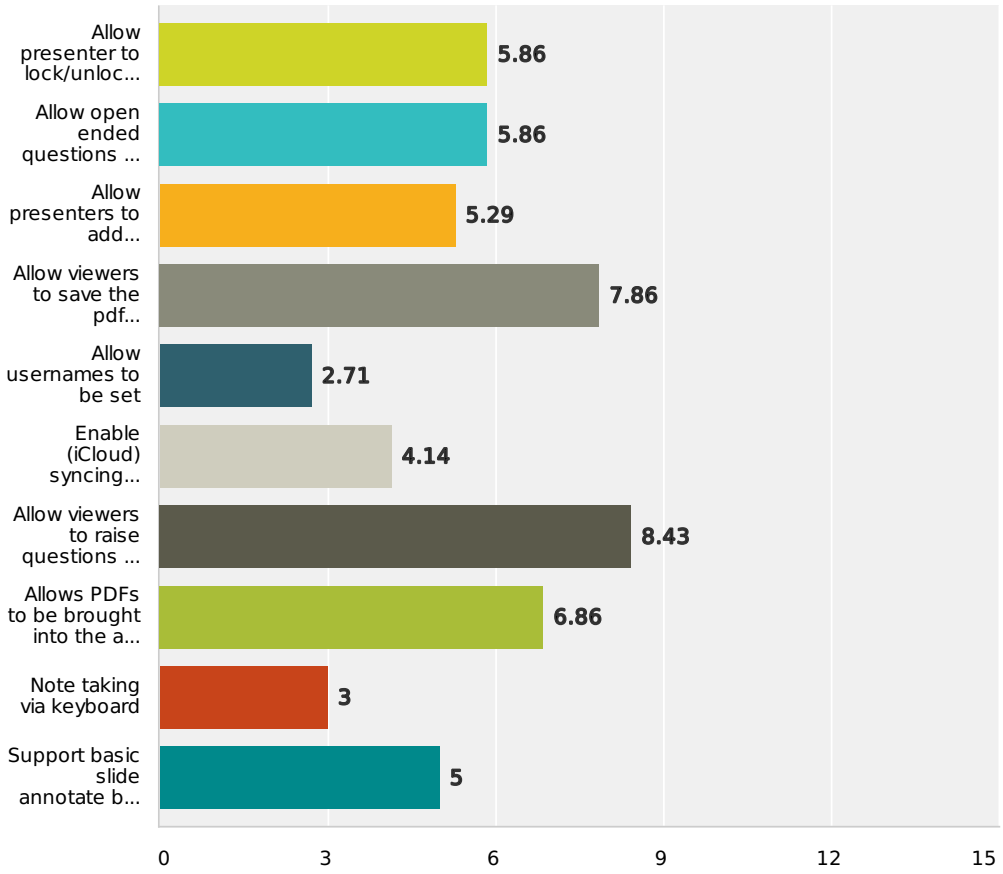
Q3 What do you think could be improved in the interface?

Answered: 7 Skipped: 0

#	Responses	Date
1	Questions and different graphs, or percentages. Or as people answered the graphs move. Or you see how many people have already answered. You could see how many people are using it. A time scale for questions.	3/9/2013 7:22 PM
2	rethink main function toolbar to make it more consistent with iPhone interface and to reduce jarring effects on rotation of the iPad	12/11/2012 10:50 AM
3	Settings and Import are smaller than User Guide - if the app is as intuitive does user guide really need such a large button?	12/11/2012 9:32 AM
4	Not sure.	12/9/2012 7:48 AM
5	Smaller/more distinct app name/logo, currently looks too similar to the other functions below? The proportions of the buttons - defining the functions in each may help. Perhaps the user guide is not that necessary? Evernote style start screen to highlight the functions e.i : broadcast - present - question	12/7/2012 3:23 PM
6	Everyone to br able to highlight/use pointer, so you can highlight an area/question to the presenter.	12/7/2012 3:23 PM
7	-the Leave the broadcast buttons should be easier to press	12/7/2012 3:08 PM

Q4 What additional features do you think should be implemented? (please rank the features below. 1. highest)

Answered: 7 Skipped: 0



	1	2	3	4	5	6	7	8	9	10	Total	Average Ranking
Allow presenter to lock/unlock the viewers screen (allowing presenters to ensure that viewers are on the correct slide).	**0%** 0	**14.29%** 1	**28.57%** 2	**14.29%** 1	**0%** 0	**0%** 0	**28.57%** 2	**0%** 0	**0%** 0	**14.29%** 1	7	5.86
Allow open ended questions to be sent out (where users can enter text based replies).	**0%** 0	**0%** 0	**0%** 0	**57.14%** 4	**14.29%** 1	**14.29%** 1	**0%** 0	**0%** 0	**14.29%** 1	**0%** 0	7	5.86

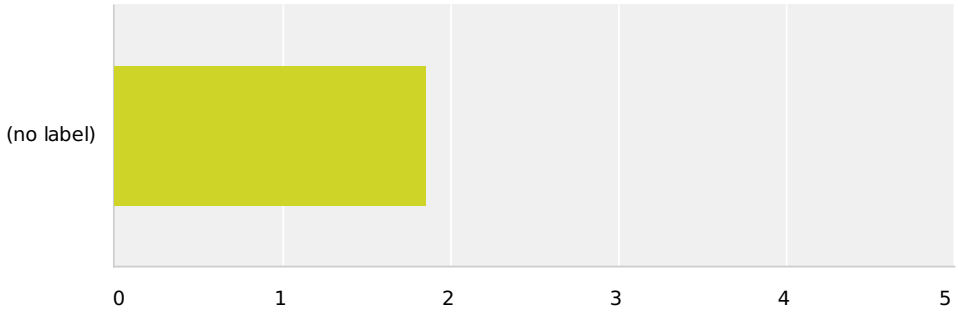
Q5 Any other feature suggestions?

Answered: 6 Skipped: 1

#	Responses	Date
1	Could it be possible for the viewers to also highlight things if they were asking a question.	3/9/2013 7:22 PM
2	Some more animation effects when a peer device connects to share its presentations.	12/11/2012 10:50 AM
3	Have some analytics features so admins can see who has viewed and saved the slides - could be used as lecture attendance	12/11/2012 9:32 AM
4	Not sure.	12/9/2012 7:48 AM
5	More interaction between the user and presenter. Ability for several presenters to combine material? As in in a presentation if someone said "I have a document that links to this" or "what about research by such and such", making it more shared than presenter vs user.	12/7/2012 3:23 PM
6	-Author/presenter name displayed on PDFs -Allow viewers to annotate slide when summoned -running order for events with different speakers	12/7/2012 3:08 PM

Q6 How fast did the application run on your device?

Answered: 7 Skipped: 0



	very fast	fast	average	slow	very slow	Total	Average Rating
(no label)	**14.29%** 1	**85.71%** 6	**0%** 0	**0%** 0	**0%** 0	7	1.86

Q7 What did you like best about the application?

Answered: 7 Skipped: 0

#	Responses	Date
1	Highlighting tool. Possibility to ask questions.	3/9/2013 7:22 PM
2	Visual layout and feel - felt like a really nice hybrid functional presentation tool and magazine app. A+ job	12/11/2012 10:50 AM
3	fast, responsive and useful. Would be of benefit and would mean people couldnt read ahead in the slides. The questions feature was my favourite - it is much better than using current handsets for quizzes	12/11/2012 9:32 AM
4	Actions on the broadcasting device are replicated on the other devices. The pointer is especially useful in this respect.	12/9/2012 7:48 AM
5	Question feature, removes the stigma from presentations. The zoom feature and laser pointer helps.	12/7/2012 3:23 PM
6	Having to be on the same slide as the presenter, I find this very useful.	12/7/2012 3:23 PM
7	Simple interface	12/7/2012 3:08 PM

Q8 What do you think could be improved with this application? (e.g did you find anything confusing?)

Answered: 6 Skipped: 1

#	Responses	Date
1	Locking of slides. Making your own annotations that could be saved.	3/9/2013 7:22 PM
2	The interactive presenter button top left in the applications main toolbar.	12/11/2012 10:50 AM
3	Closing the app to multitask was an issue - this should be fixed.	12/11/2012 9:32 AM
4	Allow the user to save the pdf and raise questions.	12/9/2012 7:48 AM
5	No.	12/7/2012 3:23 PM
6	Pointer doesnt fit with look of app	12/7/2012 3:08 PM

Appendix G

Code listings

```
- (void)showLaserPointerAtAbsolutePoint:(CGPoint)point{
    point.y-=50; // so above finger
    if (!self.laserPointerImageView) { // lazy loading - showing for first time
        UIImage *image = [UIImage imageNamed:@"laserPointer.png"];
        self.laserPointerImageView = [[UIImageView alloc] initWithImage:image] ;
        int imageWidthHeight = deviceIsIpad() ? image.size.width : image.size.width /2; // use smaller laser for iPhone
        self.laserPointerImageView.frame = CGRectMake(point.x - image.size.width/2, point.y, imageWidthHeight, imageWidthHeight);
        self.laserPointerImageView.alpha=0.8;
        [self.view addSubview:self.laserPointerImageView];
    } else if (self.laserPointerImageView.alpha==0.0){ // laser reappearing
        self.laserPointerImageView.alpha=0.8; // allow the document content to partly show behind the laser
        [self.laserPointerImageView setFrame:newFrame];
    } ...
```

Figure G.1: Method invoked by the long tap gesture recogniser to display the laser point

```
+ (PDF *) PDFWithStringPath:(NSString *)path inManagedObjectContext:(NSManagedObjectContext *) context{
    PDF *pdf = nil;
    NSFetchRequest *request = [NSFetchRequest fetchRequestWithEntityName:@"PDF"]; // first check if PDF already exists
    request.predicate = [NSPredicate predicateWithFormat:@"filePath == %@", path];
    request.sortDescriptors = [NSArray arrayWithObject:[NSSortDescriptor sortDescriptorWithKey:@"title" ascending:YES]];
    NSArray *matches = [context executeFetchRequest:request error:NULL];
    if (!matches || [matches count]>1){ //some error .. db can not be accessed or there is more than 1 match }
    else if ([matches count] ==0){ // add PDF to database
        pdf = [NSEntityDescription insertNewObjectForEntityForName:@"PDF" inManagedObjectContext:context]; // add to database
        pdf.filePath = path; // set path on entity in database
        pdf.title = [[path lastPathComponent] stringByDeletingPathExtension]; // set title on entity in database
    ...
}
```

Figure G.2: Method to insert a PDF into the data model. This method is implemented in the PDF+Path category

```
- (void)drawRect:(CGRect)rect{ // BCBarChart Drawing code
    [ self calculate ];
    float rectWidth = ( float )( rect . size . width-(self.numberOfBars)) / ( float ) self . numberOfBars;
    int barCount = 0;
    CGContextRef context = UIGraphicsGetCurrentContext();
    for (int i = 0; i < [_bars count]; i++) { // for each bar
        UIColor *colour = [ _barColours objectAtIndex :( i % _barColours.count) ];
        CGContextSetFillColorWithColor(context, colour.CGColor);
        float iLen = [[ _bars objectAtIndex:i] floatValue ];
        float x = barCount * (rectWidth);
        float height = iLen * ( rect . size . height ) / _largestBar ;
        float y;
        if (height==0) height = 1;
        y = rect . size . height - height -20;
        CGRect barRect = CGRectMake(barCount + x, y, rectWidth, height);
        CGContextFillRect( context , barRect ); // draw in the rect
        barCount+=1;
        if (iLen>0){ // add value label
            ...
            CGRect valueLabel = CGRectMake(barCount+x,y,rectWidth,20);
            CGContextSetFillColorWithColor(context, [UIColor whiteColor ]. CGColor);
            [[ NSString stringWithFormat:@"%.f",iLen] drawInRect:valueLabel withFont: ... lineBreakMode:... alignment :...];
        }
        // plot x axis bar label
        CGRect barLabel = CGRectMake(barCount+x,rect.size.height-20,rectWidth,20);
        CGContextSetFillColorWithColor(context, [UIColor grayColor ]. CGColor);
        [[ NSString stringWithFormat:@"%c",i + 65] drawInRect:barLabel withFont: ... lineBreakMode:... alignment :...];
    }
    // .. ANIMATE
}
```

Figure G.3: Custom BarChart drawing code

```
        <string>PDF</string>
        <key>LSHandlerRank</key>
        <string>Alternate</string>
        <key>LSItemContentTypes</key>
        <array>
            <string>com.adobe.pdf</string>
        </array>
```

Figure G.4: Snippet from plist file to allow PDF files to be brought into the application using ‘Open In’

```
-(void) initialize { // called when view first loads
    ...
    [ self .pageControl addTarget: self action:@selector(pageAction:) forControlEvents: UIControlEventValueChanged ]; // register for events from the page
        control .
}
-(void)pageAction:(UIPageControl*)control{
    int page = _pageControl.currentPage; // page will be +1 if user tapped on right side of page control or -1 for the left
    self .pageControl. currentPage = page; // set the page indicator to the new page
    CGRect frame = self .imageScrollView.frame; // get the current frame visible
    frame. origin .x = ( self .imageScrollView.frame. size .width * page); // set frame to next page
    [ self .imageScrollView scrollRectToVisible :frame animated:YES];
}
```

Figure G.5: Additonal code added to AFImageViewer to allow users to change page by interacting with the UIPageControl

Appendix H

Usability Evaluation

H.1 Scoring SUS

SUS yields a single number representing a composite measure of the overall usability of the system being studied. Note that scores for individual items are not meaningful on their own.

To calculate the SUS score, first sum the score contributions from each item. Each item's score contribution will range from 0 to 4. For items 1,3,5,7,and 9 the score contribution is the scale position minus 1. For items 2,4,6,8 and 10, the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SU. SUS scores have a range of 0 to 100.

H.2 Task Sheet

Task sheet used for the 'think aloud' evaluation discussed in chapter 5.

Interactive Presenter Evaluation

This sheet lists the tasks involved in this evaluation. Please note that you may not ask for help completing the tasks, but you may ask for clarification on what each task is asking you to do. You are encouraged to talk through your thought process as you are carrying out these tasks, the evaluator will take notes based on this.
Remember it is the application, not you, that is being tested.

Please read through the in app User Guide. *This should take 3-5 mins.*

Presenter

For these tasks you will take on the role of a presenter. The demonstrator will have two iOS devices which will be used to view your presentation.

1. Import a PDF into the application from an online cloud service.

2. View the PDF you just imported. If you were unable to import a PDF into the application view one of the built in ones.

3. Add a multiple choice question to the PDF with 4 possible answers.
   E.g. "Who makes iOS?" . Apple, Blackberry, Microsoft or HP.

4. Broadcast the the PDF with the passcode: *1234* so others can start watching.
   Please inform the demonstrator when you think you have done this so they can join as a viewer.

5. Navigate through different parts of the PDF taking the role as a presenter. Use the zoom and laser pointer functionality.
   Observe the effect this has on the viewers devices.

6. Send out the multiple choice question you made in task 3.
   Please inform the demonstrator when you think you have done this so they can answer the question.

7. Reveal just the answer to the viewers.
   Observe the effect this has on the viewers devices.

8. Then reveal the answer & results.
   Observe the effect this has on the viewers devices.

9. Finish the presentation and return to the list of PDFs in the application.
   Please inform the demonstrator when you think you have done this.

Viewer

For these tasks you will take on the role of a viewer. The demonstrator will have an iOS device which act as the presenter.

10. Join the 'iOS lecture' with the passcode: *qwerty.*
    Please inform the demonstrator when you think you have done this.

11. Answer the multiple choice question sent to you by the presenter.
    Please inform the demonstrator when you think you have done this.

12. Ask the presenter a question via the application.
    Please inform the demonstrator when you think you have done this.

The presenter will now end the presentation.

13. *Carry on browsing the PDF. You will then be able to navigate freely around the presentation.*

14. Leave the presentation and save the PDF.

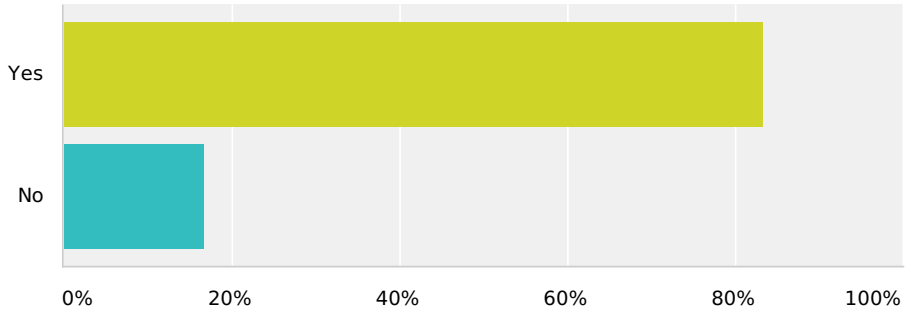
Misc

14. Invite a friend to the application via email. Use the email 0908695c@student.gla.ac.uk

H.3 Survey results

Q1 Do you currently have an iOS device?

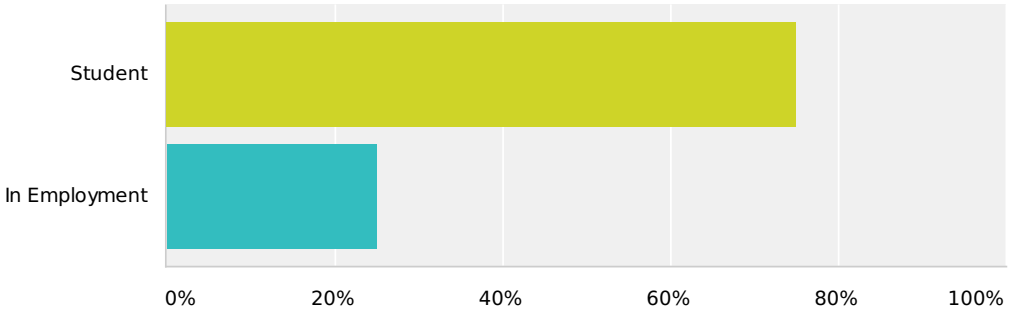
Answered: 12 Skipped: 0



Answer Choices	Responses	
Yes	**83.33%**	10
No	**16.67%**	2
Total		12

Q2 Which category do you fall under?

Answered: 12 Skipped: 0

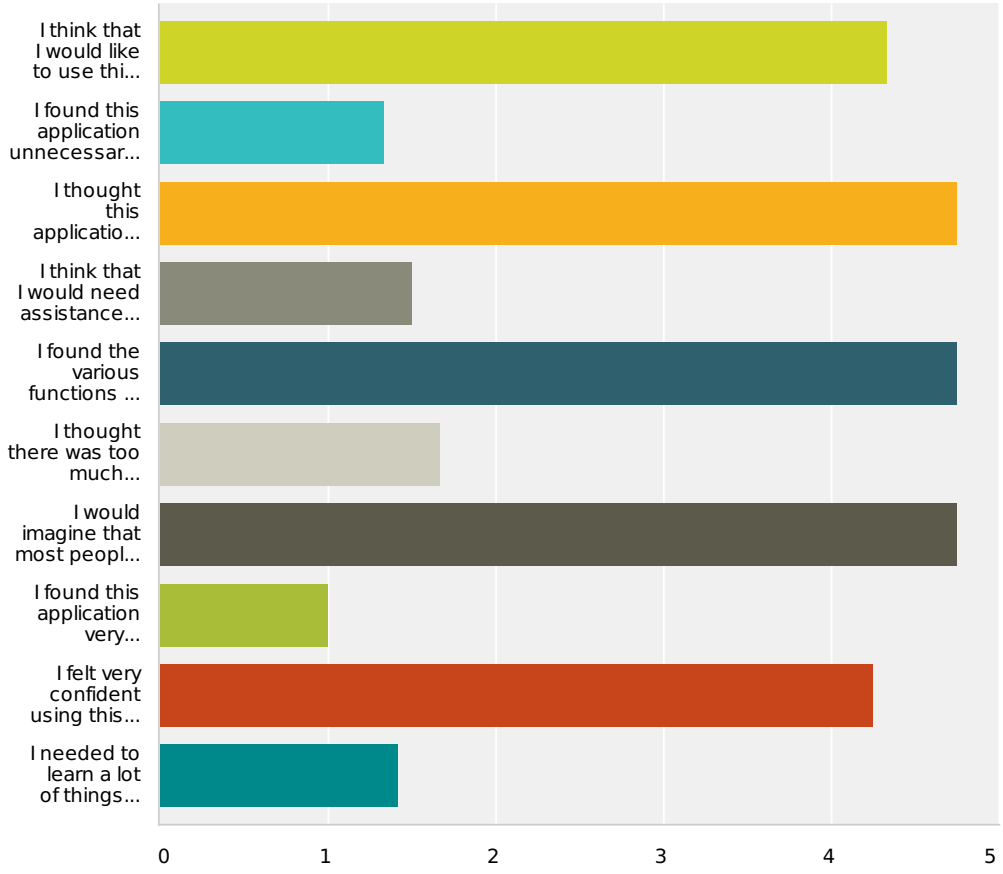


Answer Choices	Responses	
Student	**75%**	9
In Employment	**25%**	3
Total		12
Other (please specify) (0)		

#	Other (please specify)	Date
	There are no responses.	

Q3 For each of the following statements, mark one box that best describes your reactions to the application today.

Answered: 12 Skipped: 0



	Strongly Disagree	(no label)	(no label)	(no label)	Strongly Agree	Total	Average Rating
I think that I would like to use this application frequently.	**0%** 0	**0%** 0	**0%** 0	**66.67%** 8	**33.33%** 4	12	4.33
I found this application unnecessarily complex.	**83.33%** 10	**8.33%** 1	**0%** 0	**8.33%** 1	**0%** 0	12	1.33
I thought this application was easy to use.	**0%** 0	**0%** 0	**0%** 0	**25%** 3	**75%** 9	12	4.75
I think that I would need assistance to be able to use this application.	**58.33%** 7	**33.33%** 4	**8.33%** 1	**0%** 0	**0%** 0	12	1.50
I found the various functions in this application were well integrated.	**0%** 0	**0%** 0	**0%** 0	**25%** 3	**75%** 9	12	4.75

I thought there was too much inconsistency in this application.	**58.33%** 7	**25%** 3	**8.33%** 1	**8.33%** 1	**0%** 0	12	1.67
I would imagine that most people would learn to use this application very quickly.	**0%** 0	**0%** 0	**0%** 0	**25%** 3	**75%** 9	12	4.75
I found this application very cumbersome/awkward to use.	**100%** 12	**0%** 0	**0%** 0	**0%** 0	**0%** 0	12	1.00
I felt very confident using this application.	**0%** 0	**8.33%** 1	**0%** 0	**50%** 6	**41.67%** 5	12	4.25
I needed to learn a lot of things before I could get going with this application.	**58.33%** 7	**41.67%** 5	**0%** 0	**0%** 0	**0%** 0	12	1.42

Q4 What would you add to Interactive Presenter? (list multiple)

Answered: 12 Skipped: 0

#	Responses	Date
1	Audio recording of presentations.	3/12/2013 3:45 PM
2	Inviting friends from device contacts?	3/12/2013 3:33 PM
3	Loading icon when buffering pdf. Notification for when pdf downloaded from dropbox.	3/12/2013 3:15 PM
4	Help guide, so you could go back the the user guide if stuck	3/12/2013 2:54 PM
5	Ability to annotate slides	3/12/2013 7:54 AM
6	The ability to add text based notes to slides for viewer	3/11/2013 9:31 PM
7	Ability for presenter to edit PDF and for viewers to see these changes. Offline mode, so that if you miss a lecture the lecturer can upload an offline edition online that you can view on your device later.	3/6/2013 6:10 AM
8	synchronisation of files with multiple devices.	3/6/2013 5:45 AM
9	organise saved pdfs into folders	3/5/2013 5:23 AM
10	linked with something like google hangouts or similar application where you wouldnt necessarily need an iOS application to join.	3/5/2013 4:55 AM
11	-questions without correct answers	3/4/2013 8:35 AM
12	The ability to point to graphs and highlight answers. The ability to choose what type of graph is presented The ability to log out of my synced accounts so as not to have security risks.	3/4/2013 8:05 AM

Q5 What did you find difficult when using Interactive Presenter? (list multiple)

Answered: 12 Skipped: 0

#	Responses	Date
1	I was unable to find any difficulties. The help guide informed me of any questions I had when using the application.	3/12/2013 3:45 PM
2	Multiple pass code was annoying	3/12/2013 3:33 PM
3	Keeping my finger still when pressing down for pointer. This would become easier though through multiple use so not really an issue	3/12/2013 3:15 PM
4	fat fingers and small buttons	3/12/2013 2:54 PM
5	Nothing really. Once i had learnt how to do things once it wasnt hard	3/12/2013 7:54 AM
6	I wasn't aware I needed to tap the screen to get the PDF controls back	3/11/2013 9:31 PM
7	I have to be honest and say that everything was so simple.	3/6/2013 6:10 AM
8	I had to read a lot.	3/6/2013 5:45 AM
9	n/a	3/5/2013 5:23 AM
10	getting used to the different sections is difficult at first but once completed once it is easy to remember	3/5/2013 4:55 AM
11	-	3/4/2013 8:35 AM
12	There was a lack of consistence in the names of buttons and i was a bit confused as to the flow as a result of this.	3/4/2013 8:05 AM

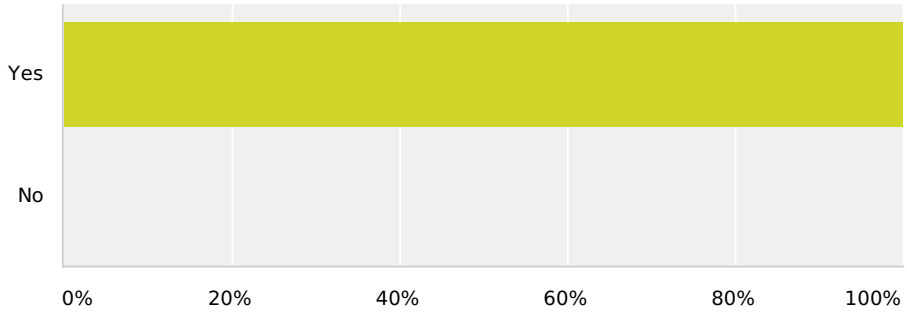
Q6 When would you imagine using Interactive Presenter?

Answered: 12 Skipped: 0

#	Responses	Date
1	I particularly liked being able to ask questions without interrupting the presenter. I can really see this being used in teaching, as often I hold back from asking a question because I would have to put my hand up and stop the teacher.	3/12/2013 3:45 PM
2	Presenting to small group of my peers casually. I could imagine this being popular in schools.	3/12/2013 3:33 PM
3	When teaching colleagues about new software or updated software with new changes.	3/12/2013 3:15 PM
4	Group meetings or revising lecture notes as a group	3/12/2013 2:54 PM
5	lectures, workshops, in business meetings, training days and in sales?	3/12/2013 7:54 AM
6	University lectures or meetings in work	3/11/2013 9:31 PM
7	Would be great to use it in a lecture.	3/6/2013 6:10 AM
8	In class of course!! At a team meeting!	3/6/2013 5:45 AM
9	More interactive lectures where the lecturer asked lots of questions. Or in a large lecture hall where you may not be able to see the board well. May be a distraction to some people though	3/5/2013 5:23 AM
10	during lectures where interactive teaching would be appropriate --- where feedback may be necessary	3/5/2013 4:55 AM
11	during lectures and could be useful for schools. it could also be used for corporate events.	3/4/2013 8:35 AM
12	In lectures or presentations at work when I graduate.	3/4/2013 8:05 AM

Q7 Would you recommend interactive presenter to somebody who was wanting to present?

Answered: 12 Skipped: 0



Answer Choices	Responses	
Yes	**100%**	12
No	**0%**	0
Total		12

Maybe (0)

#	Maybe	Date
	There are no responses.	

Appendix I

Full-size images

Figure 2.1a: Handset and receiver

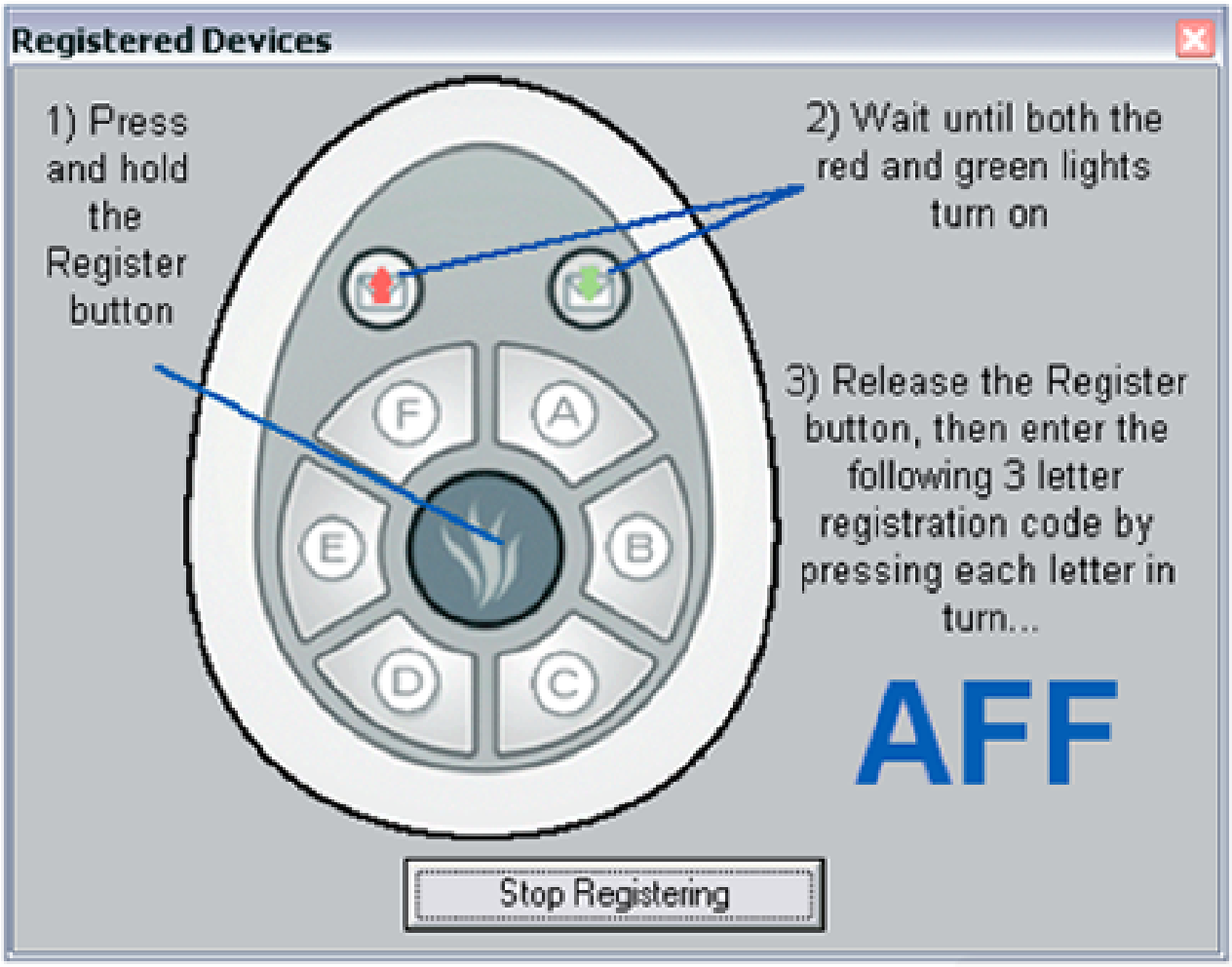


Figure 2.1b: Handset pairing

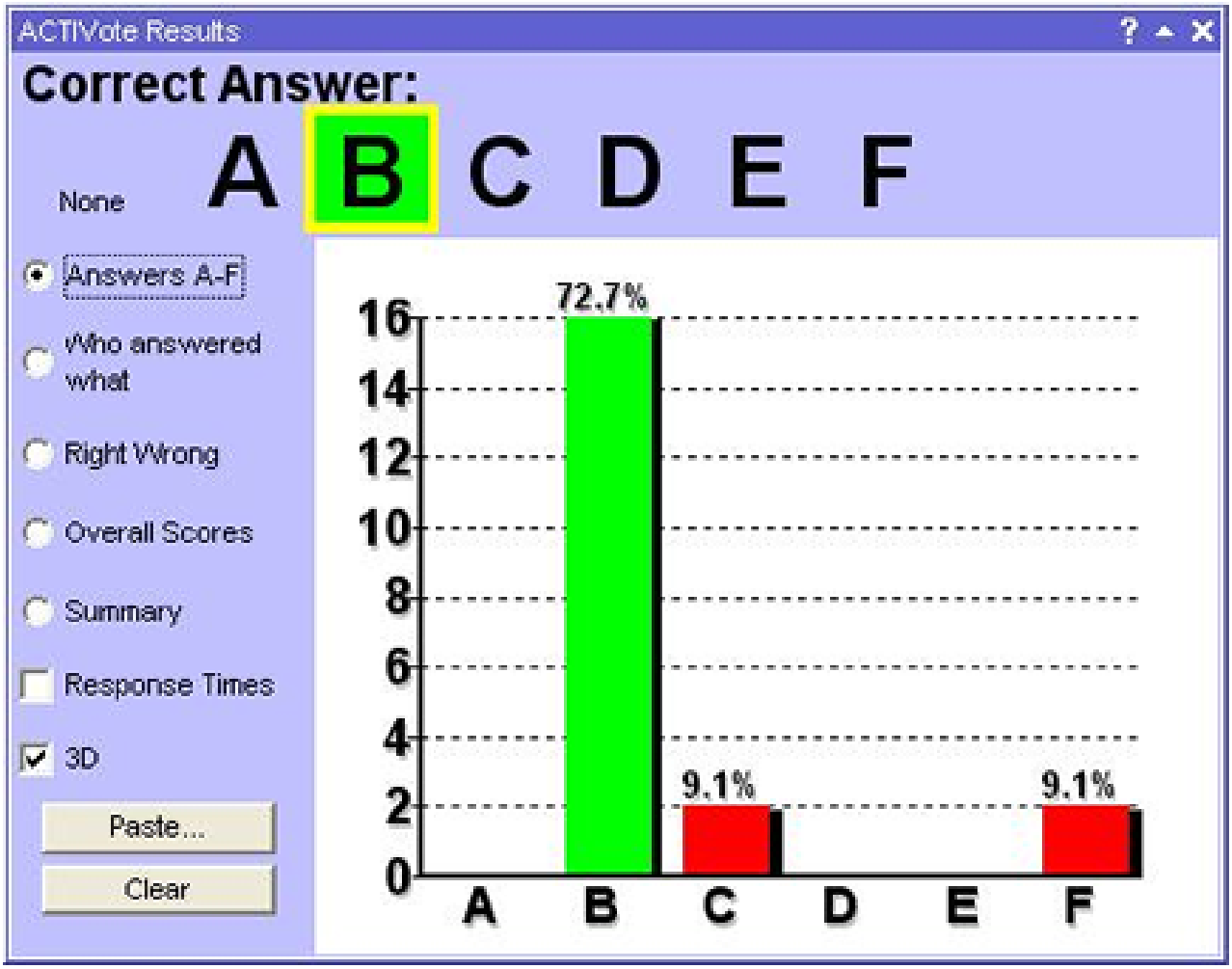


Figure 2.1c: Results screen

Figure 2.2a: Handset and receiver

Figure 2.2b: Handset buttons

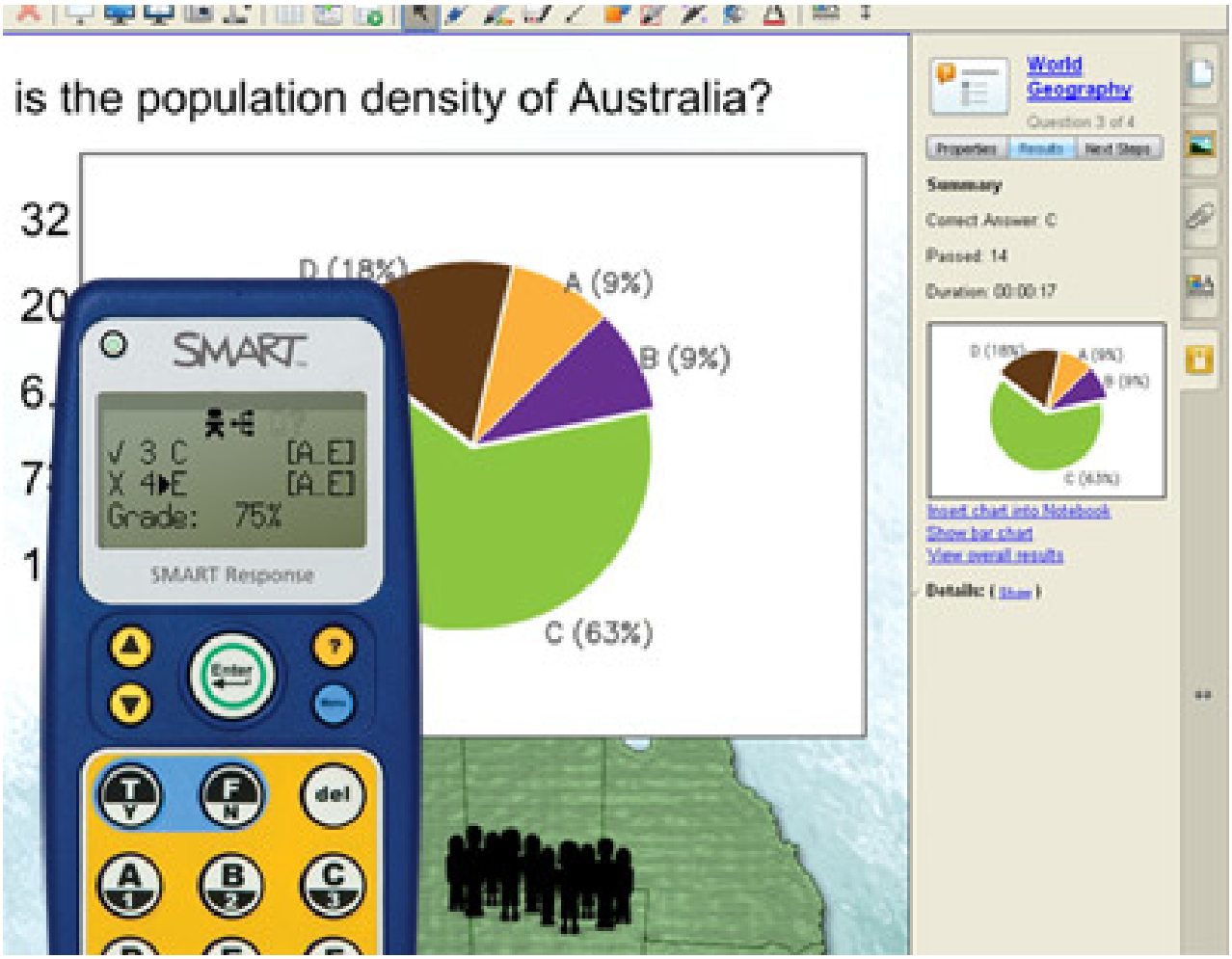


Figure 2.2c: Results screen

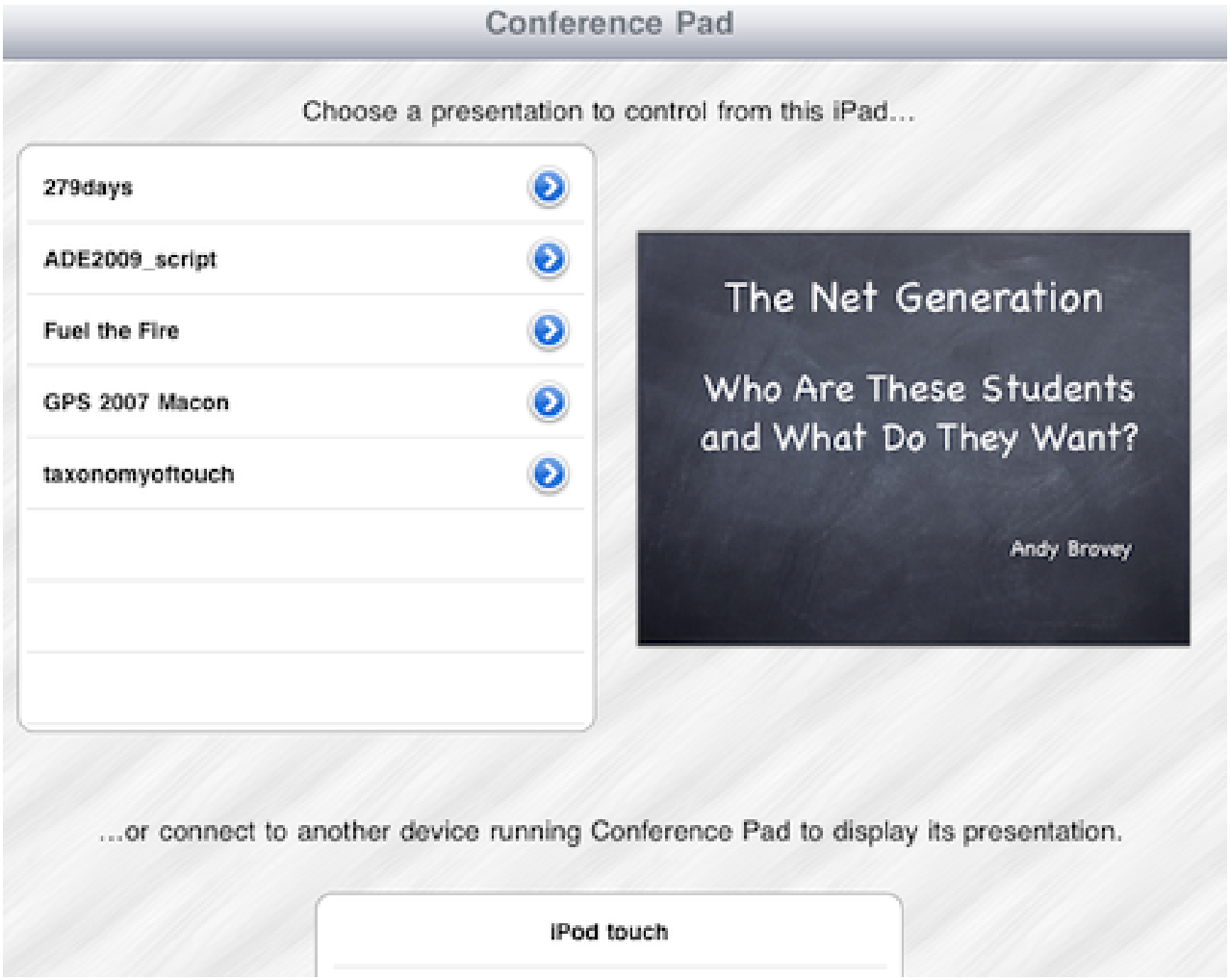


Figure 2.3a: Conference Pad

Figure 2.3b: Idea Flight

Figure 2.3c: ResponseWare

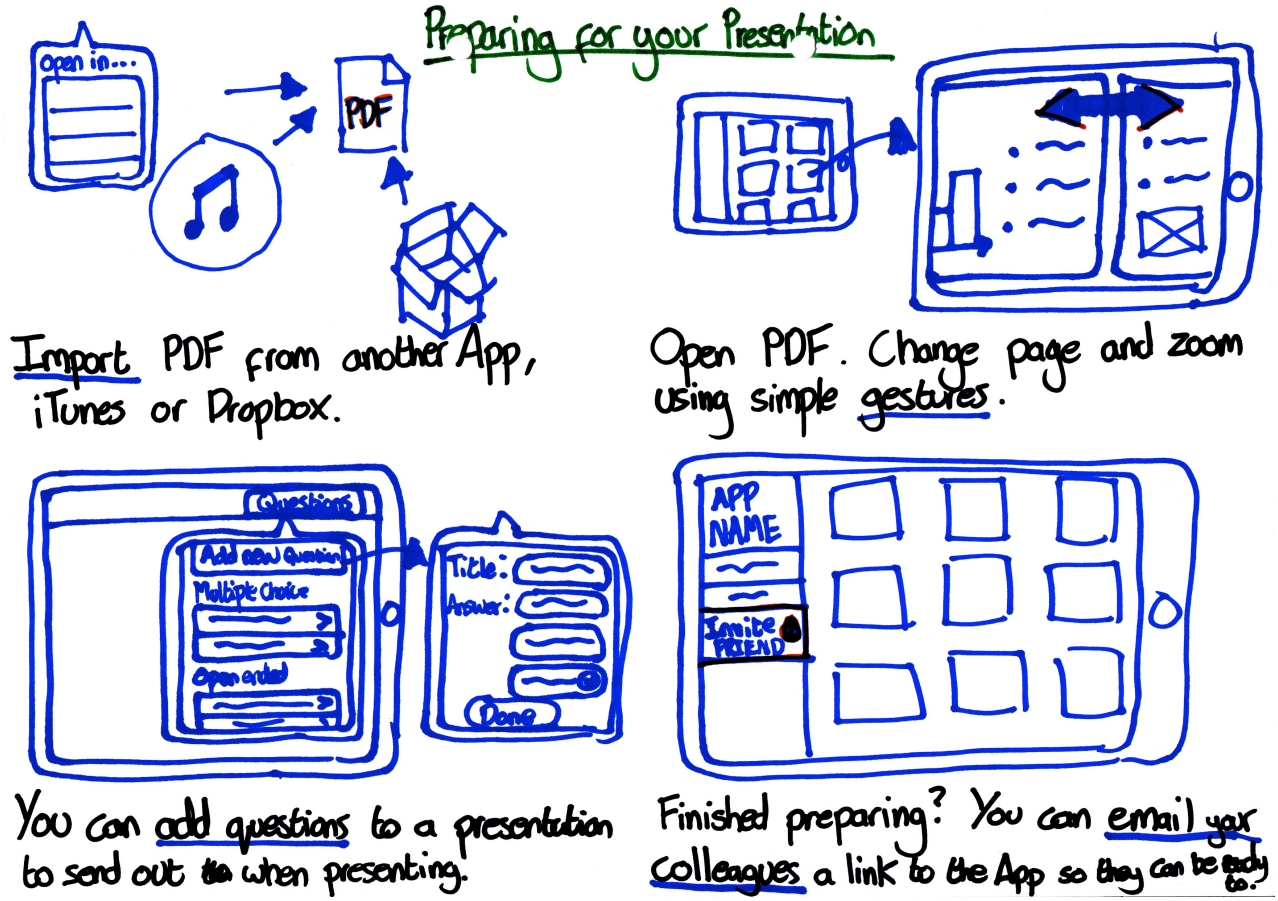


Figure 3.1: Preparing to present

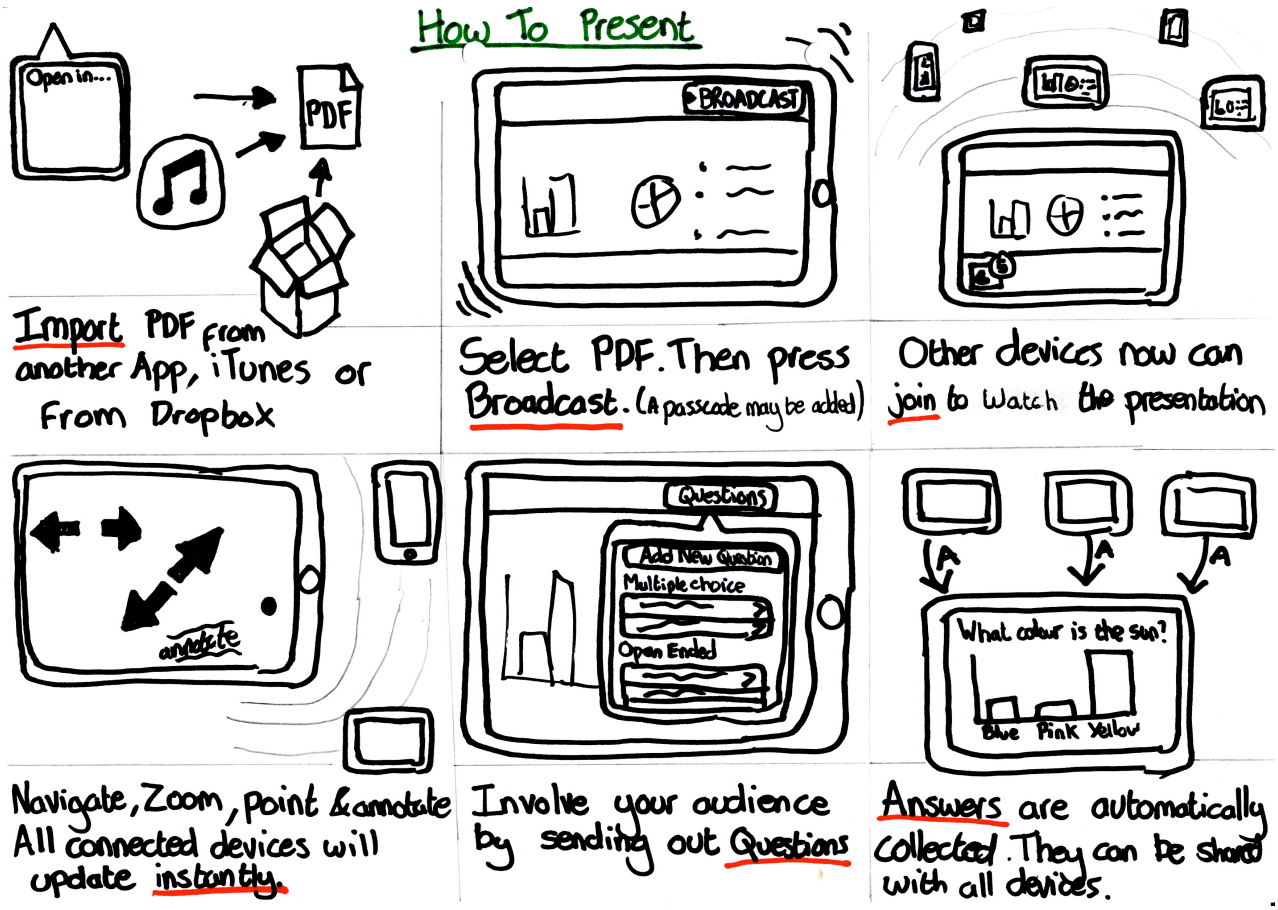


Figure 3.2: Presenting a document

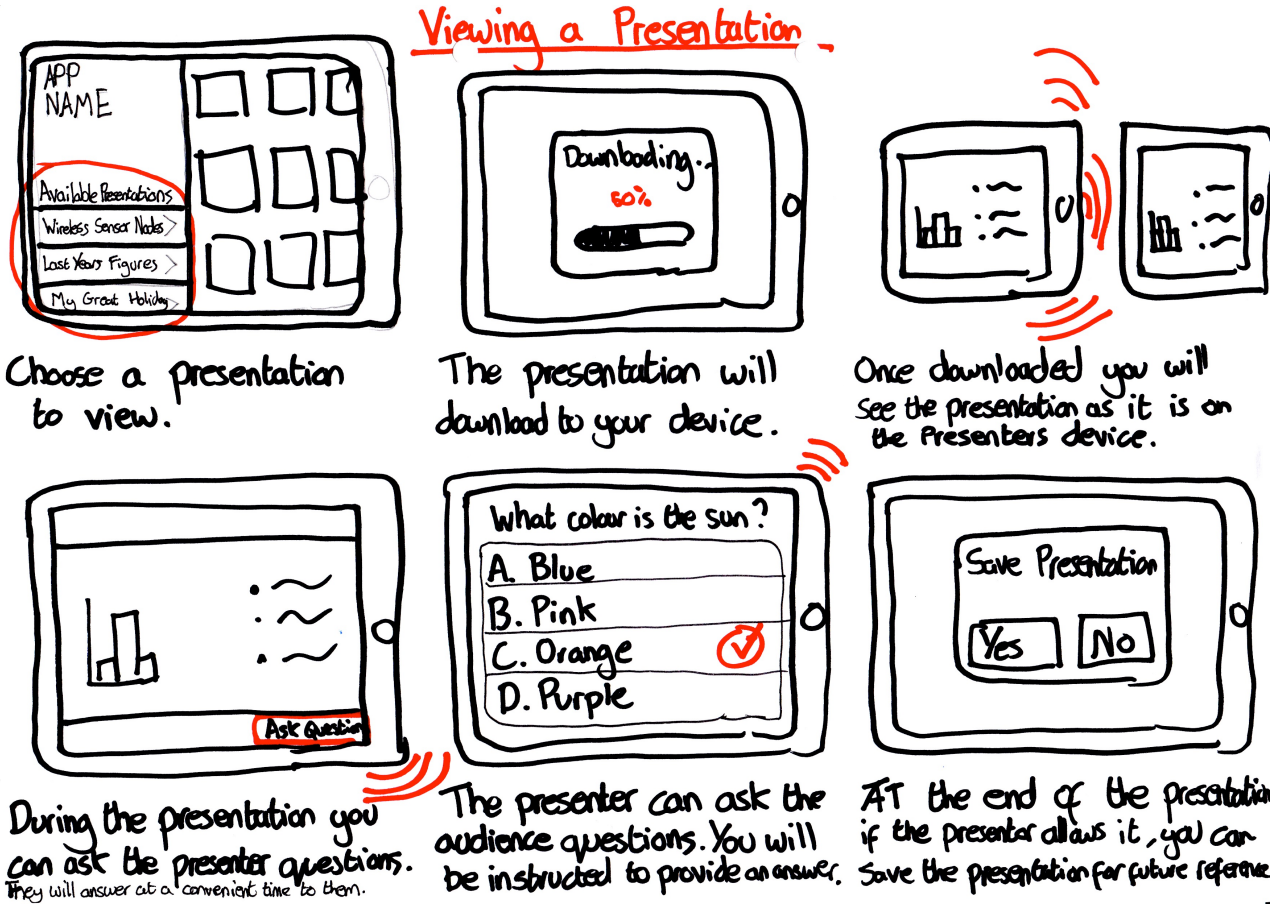


Figure 3.3: Viewing a presentation

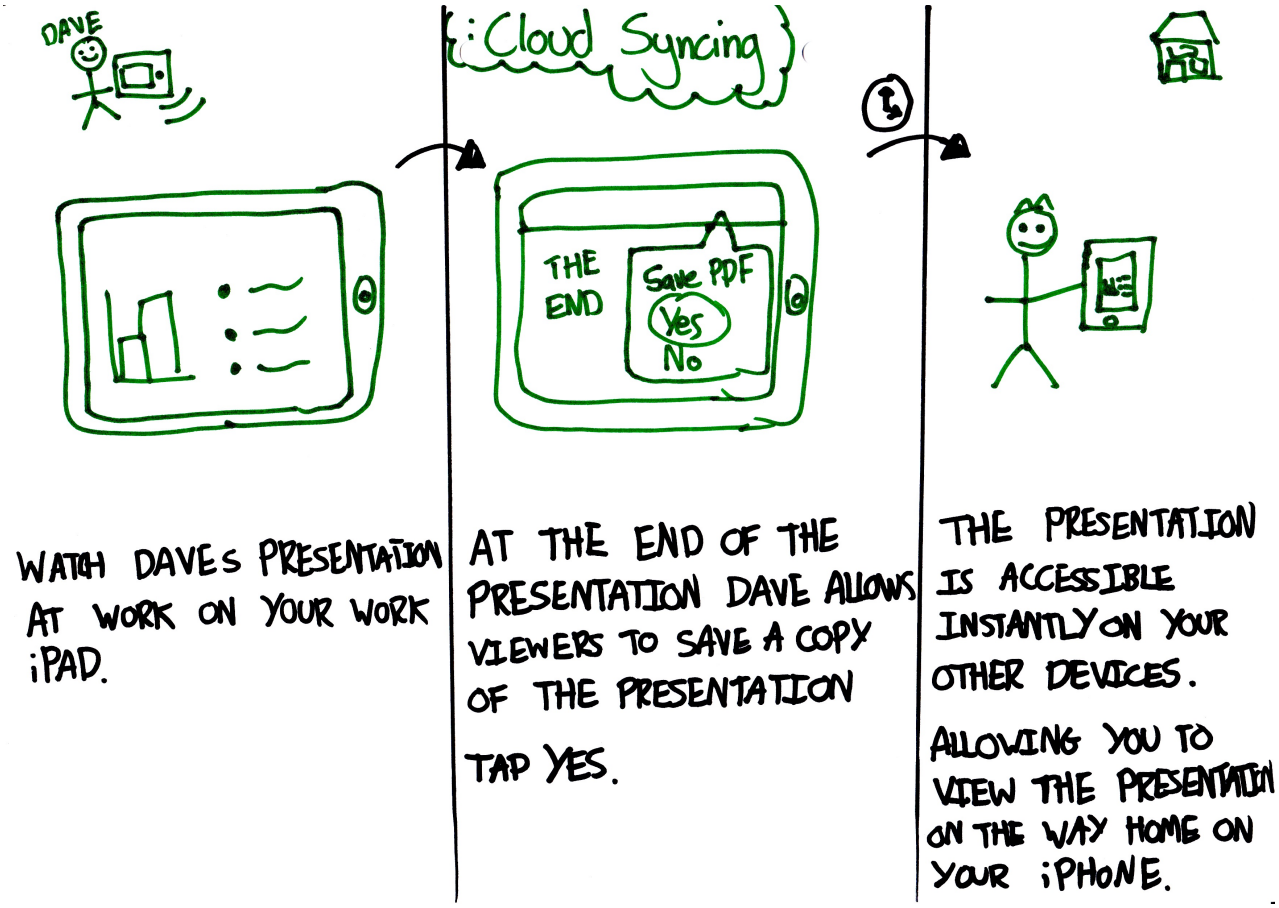


Figure 3.4: Document syncing

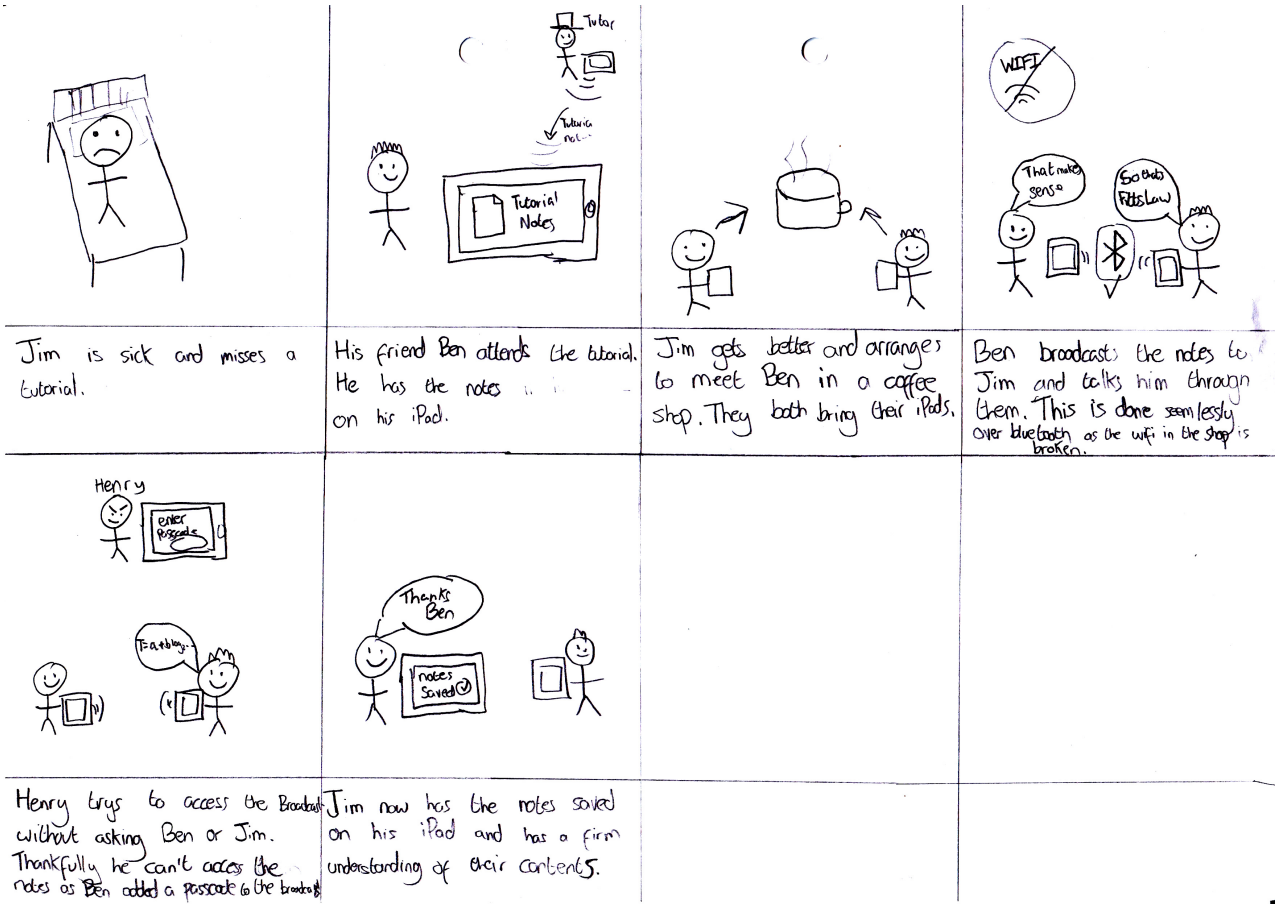


Figure 3.5a: Personal use

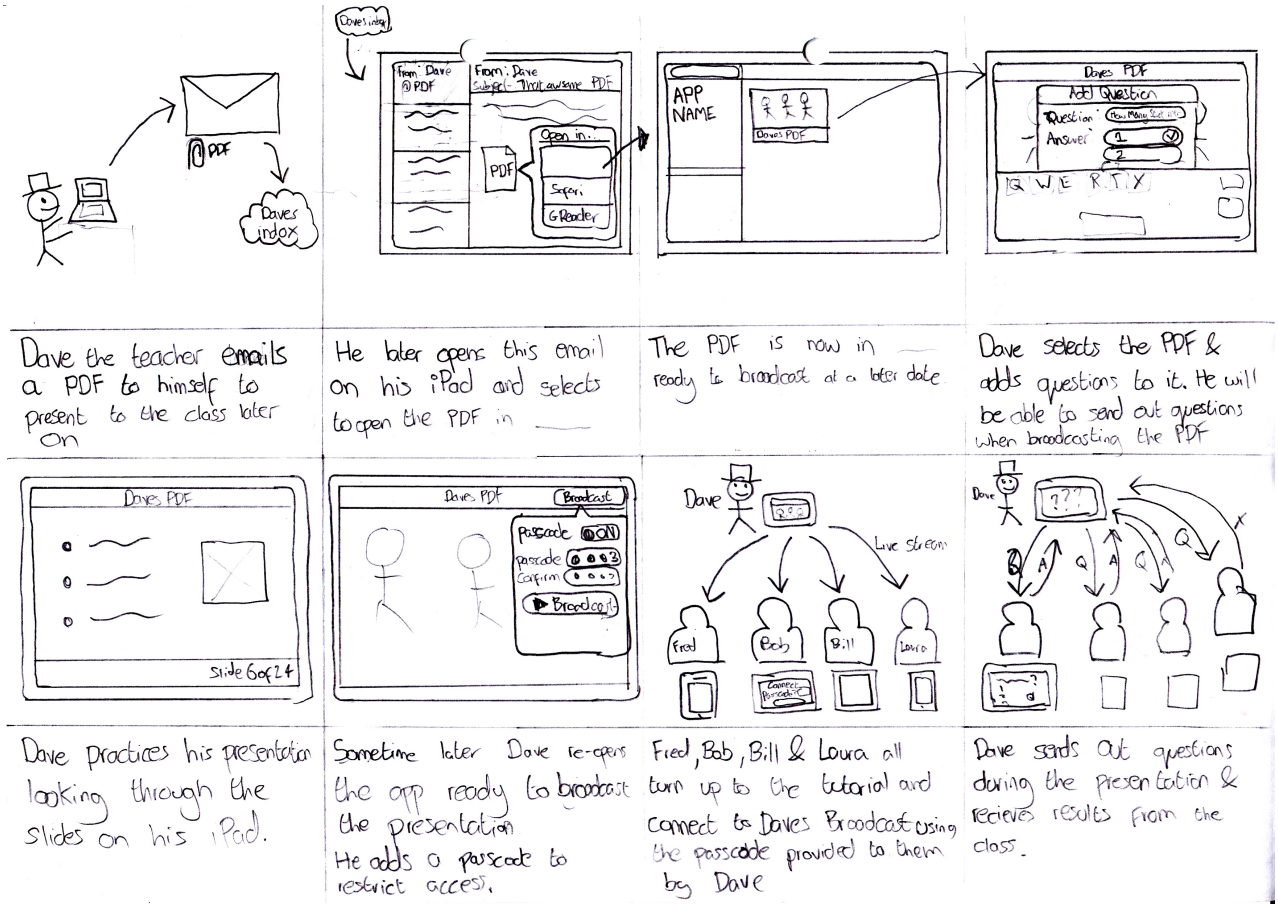


Figure 3.5b: Academic use

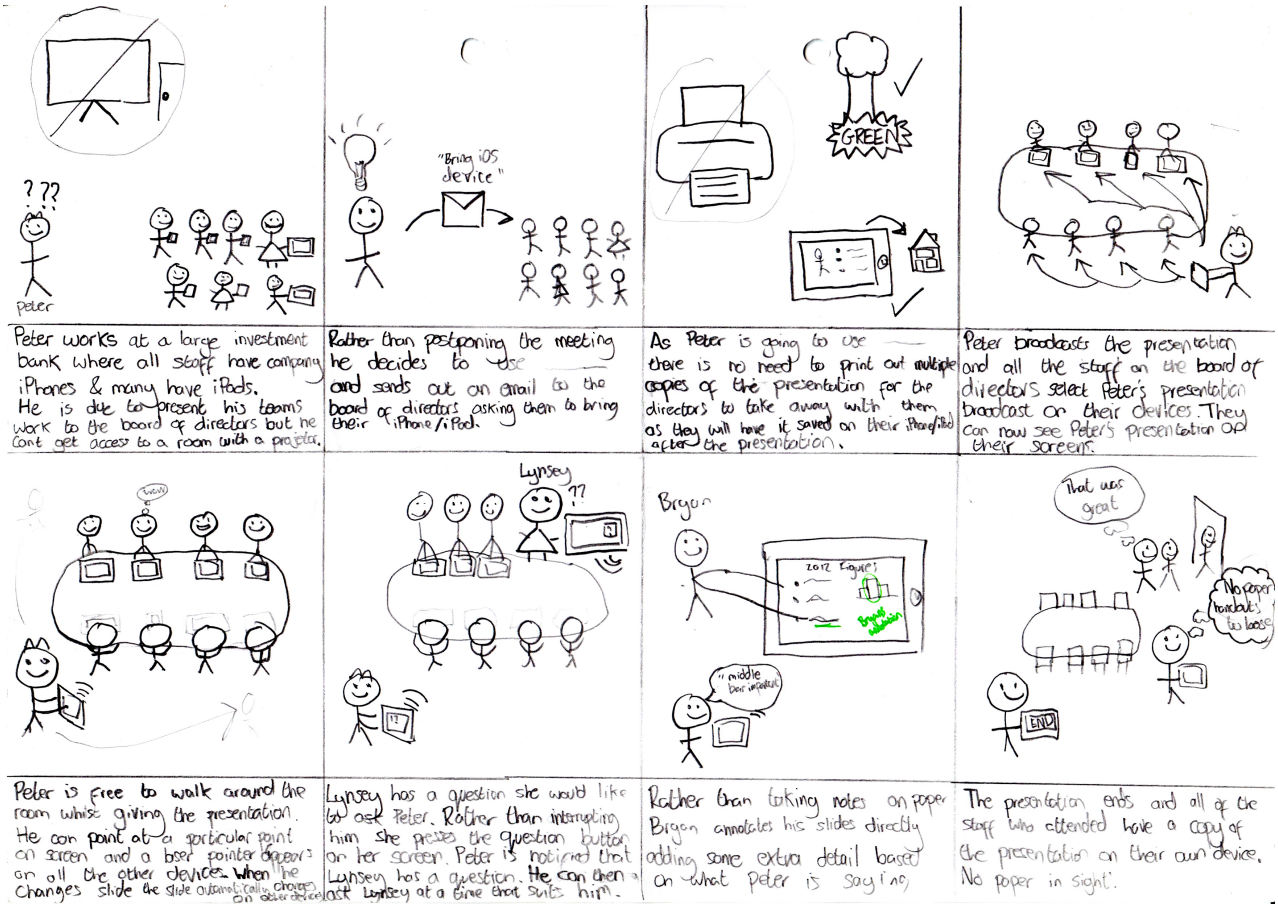


Figure 3.5c: Business use

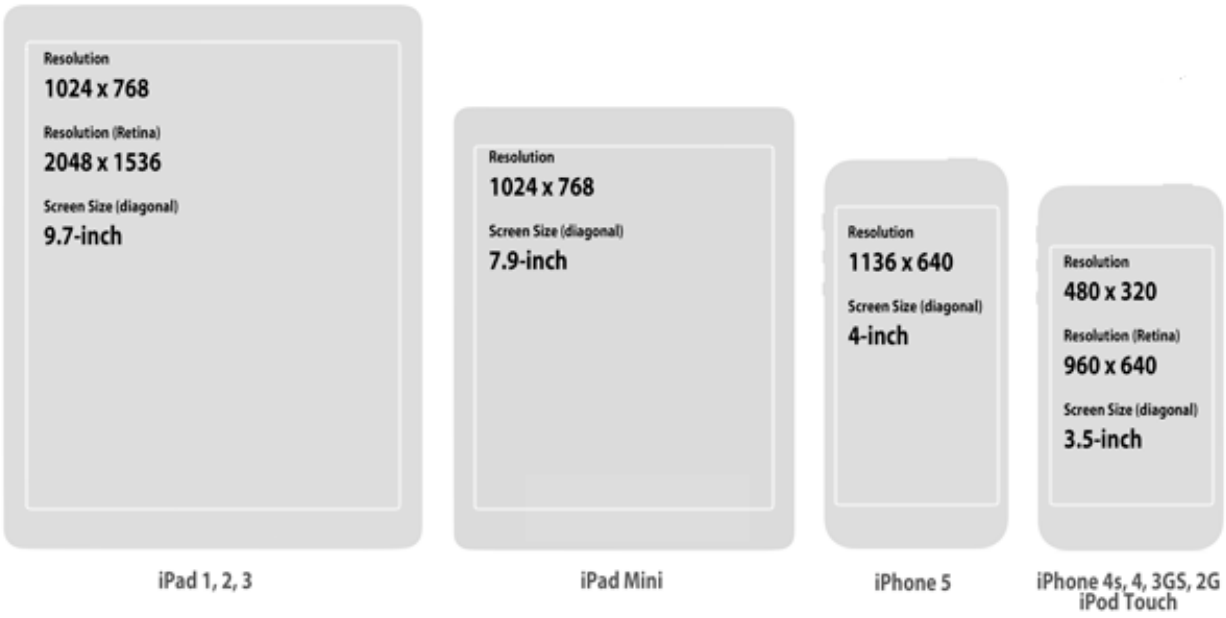


Figure 3.6a: iOS screen comparison

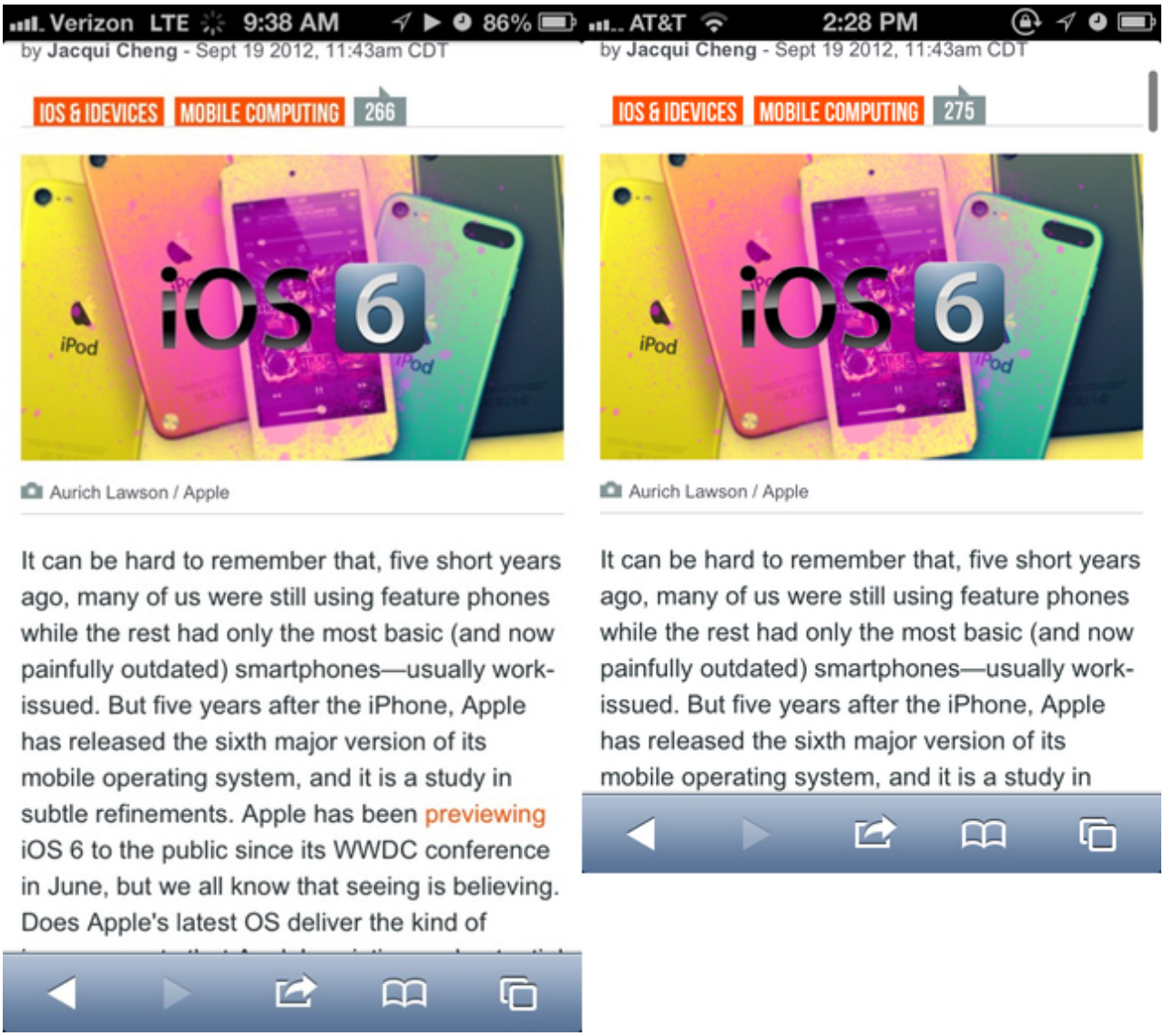


Figure 3.6b: Safari running on iPhone 5 and iPhone 4

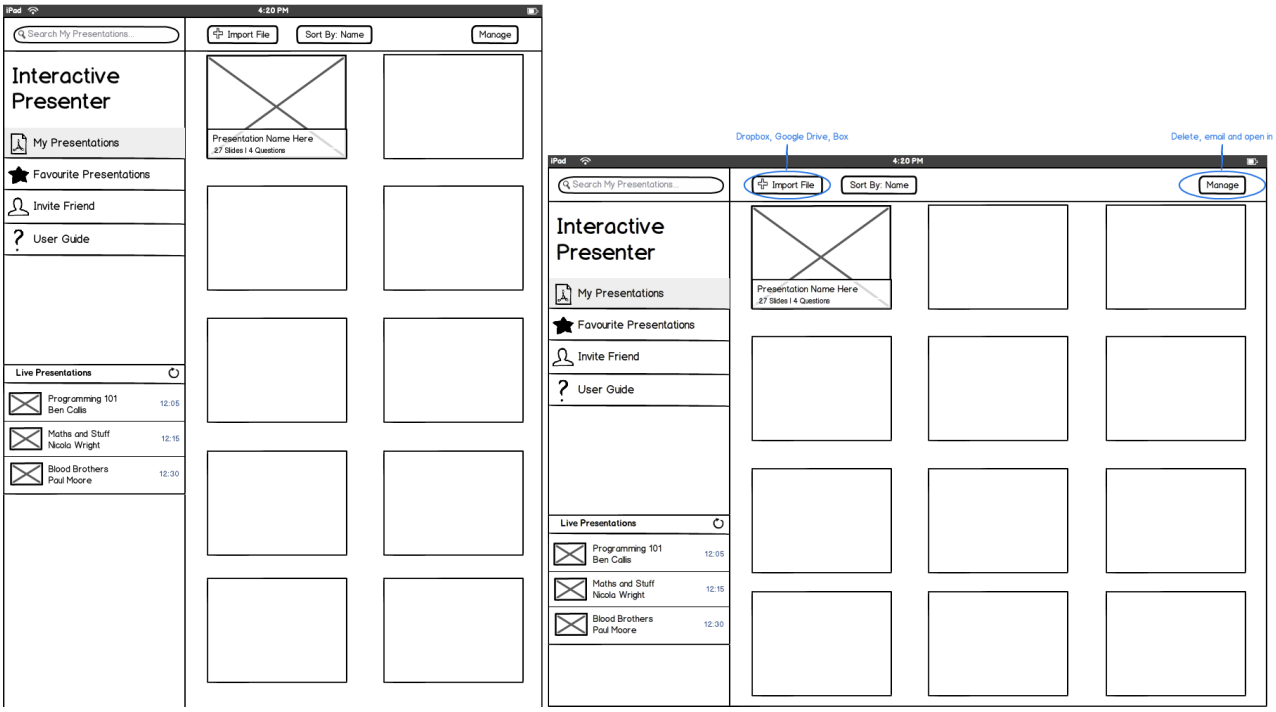


Figure 3.7a: Design 1

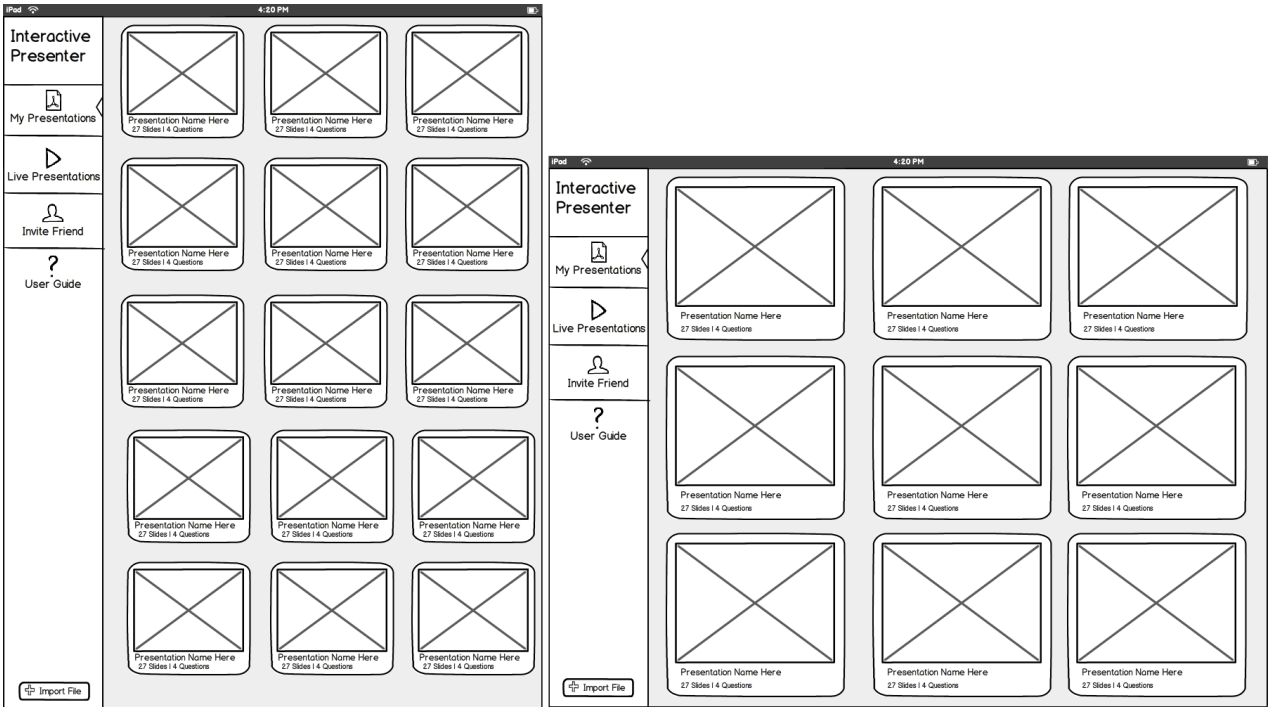


Figure 3.7b: Design 2

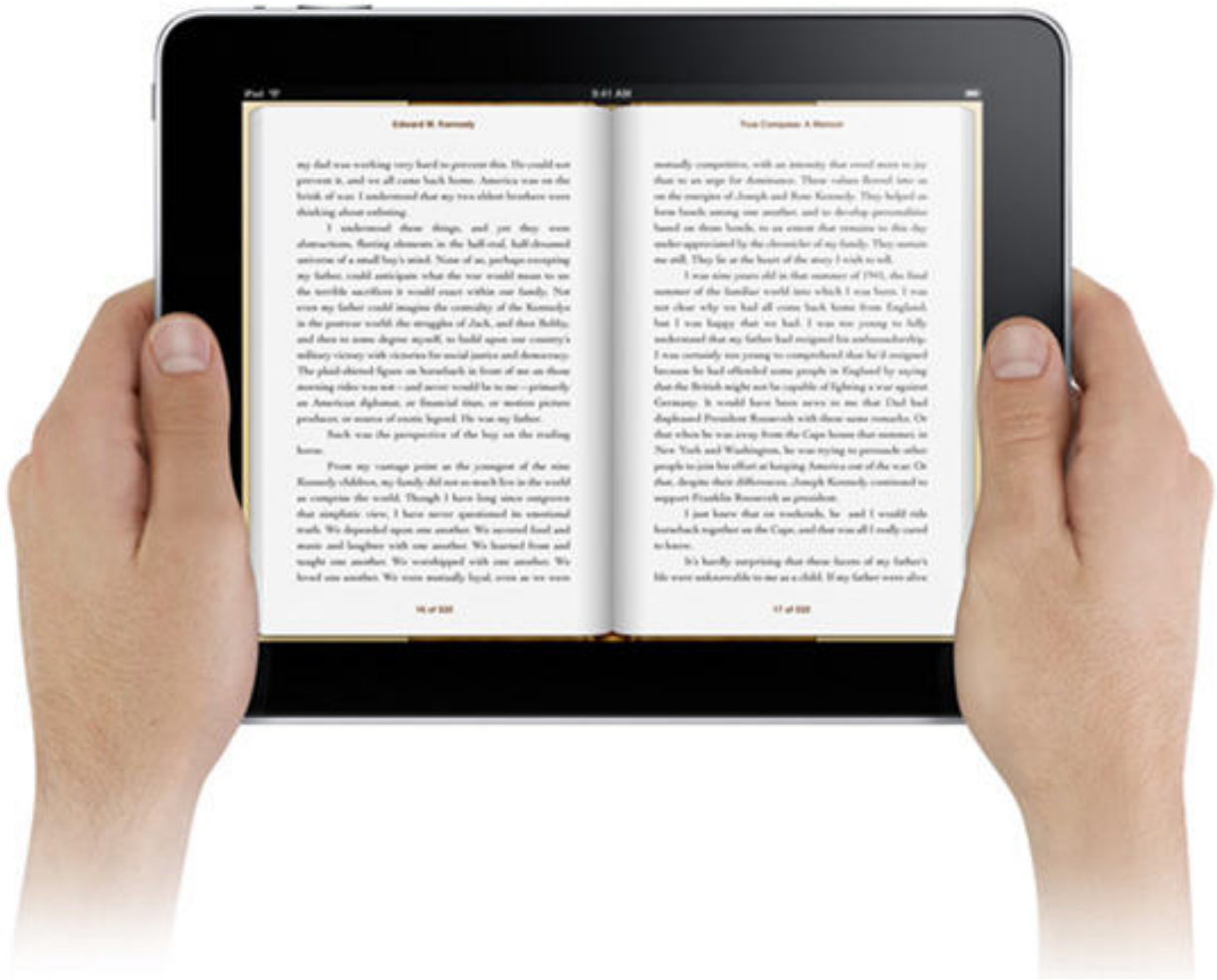


Figure 3.8a: Two hands along bezel

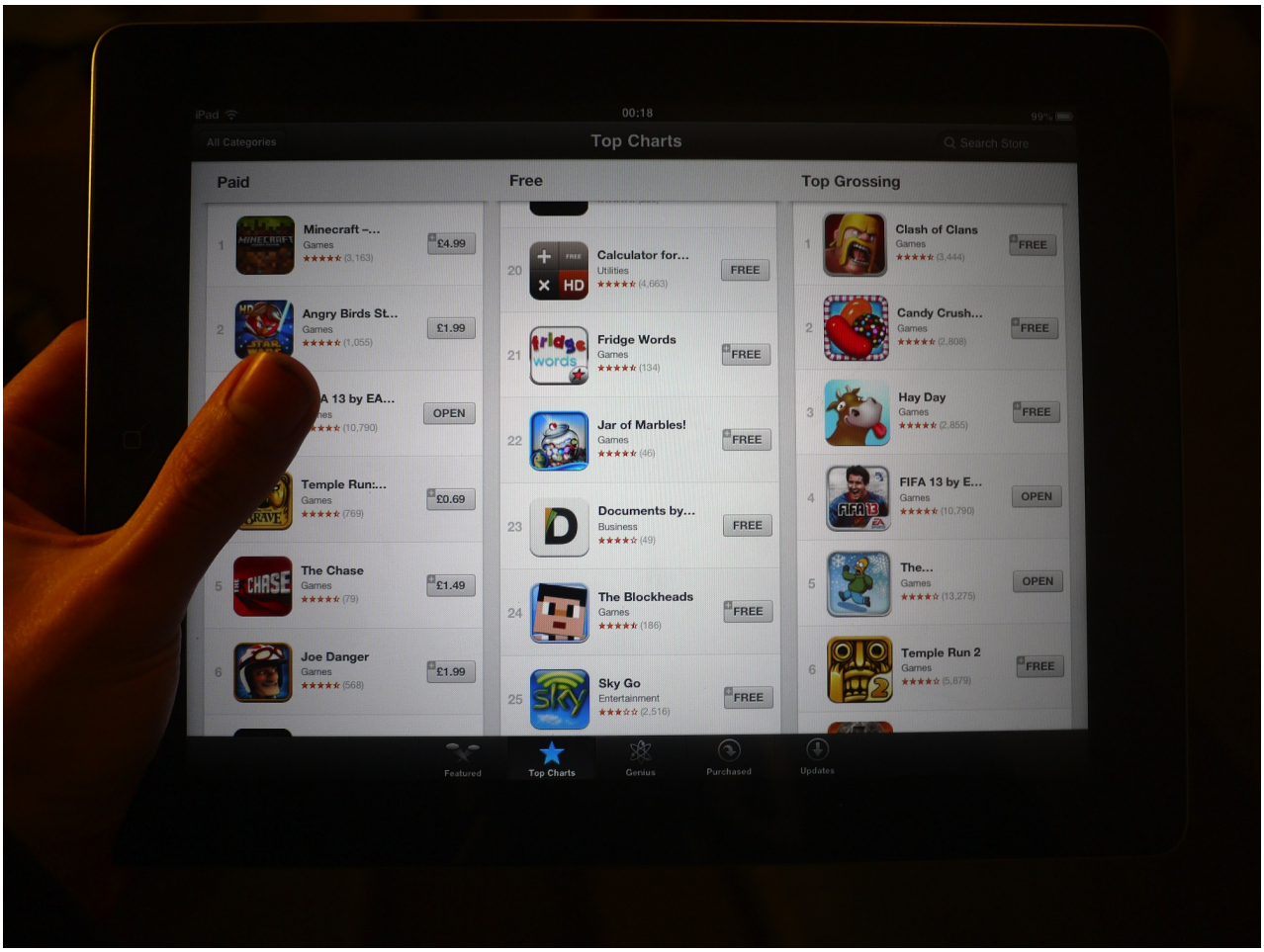


Figure 3.8b: Left hand along bezel

Figure 3.8c: Left thumb reach

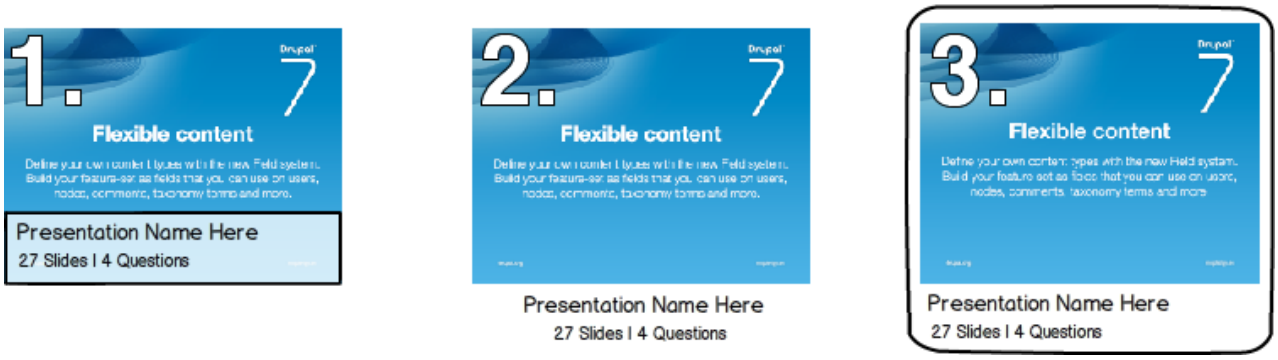


Figure 3.9: Three different cell styles used to represent a document.

Figure 3.10a: Presenter View

Figure 3.10b: Viewer view

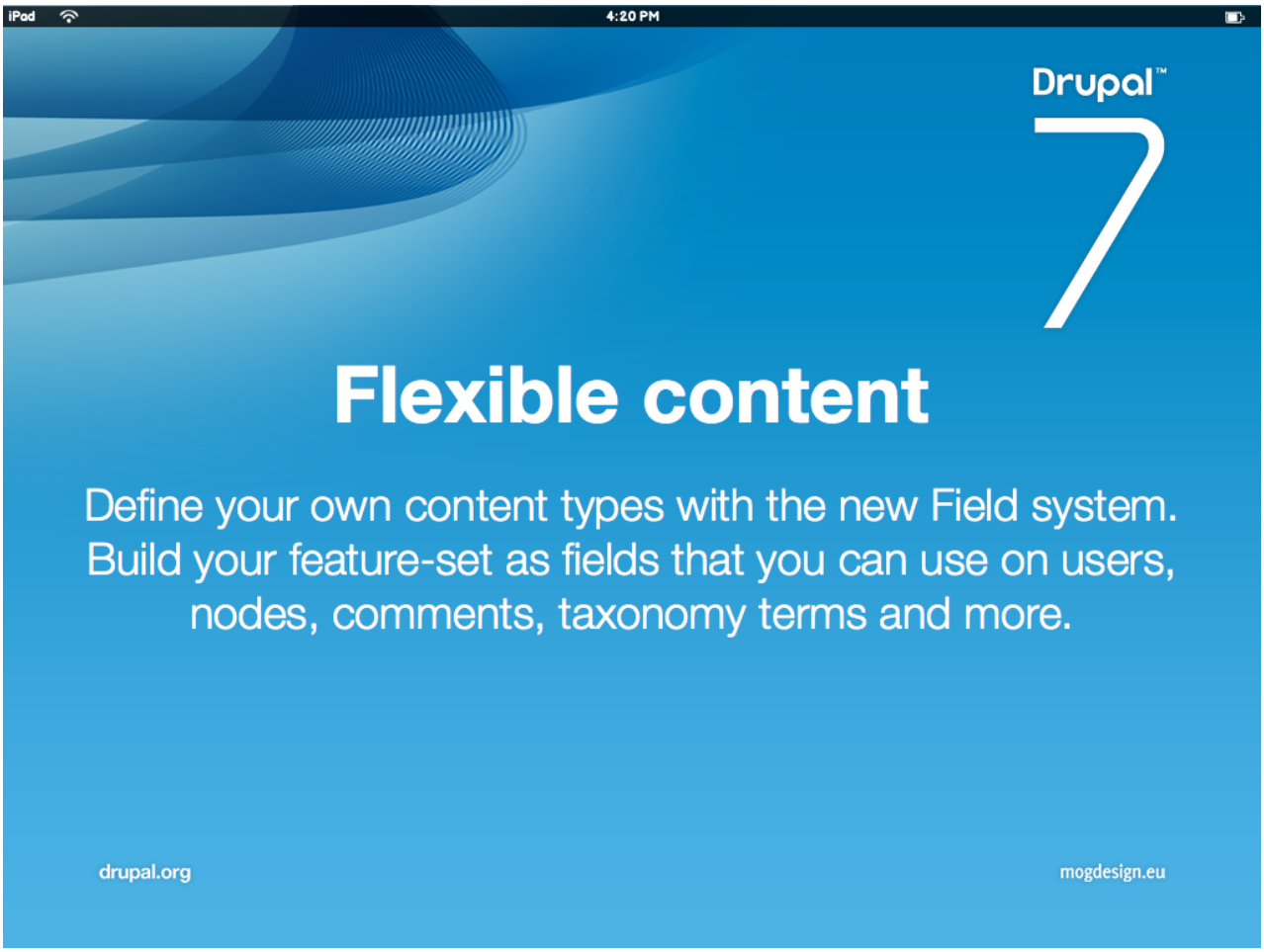


Figure 3.10c: Full screen

Figure 3.10d: Page changing

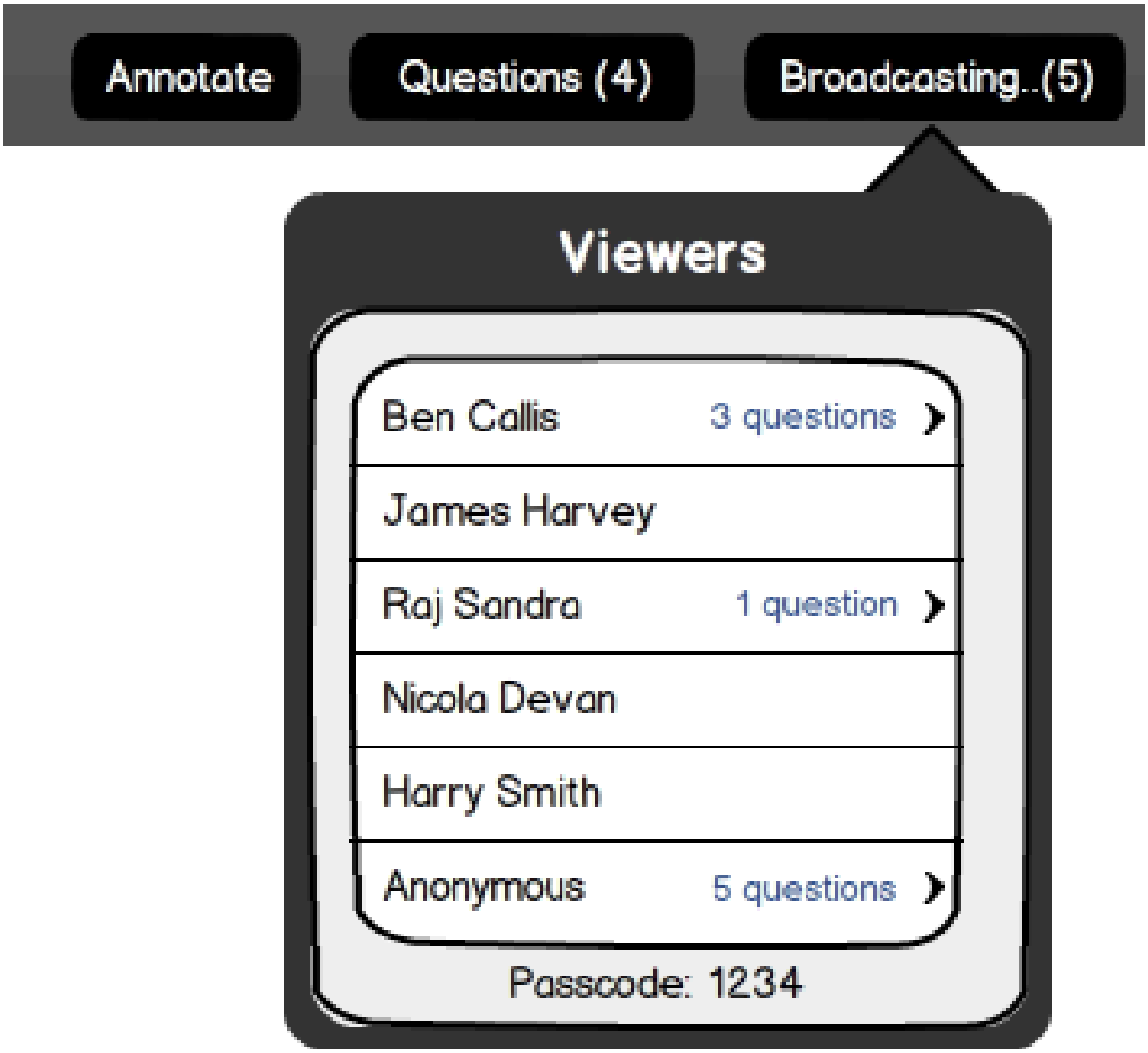


Figure 3.10e: List of viewers visible to presenter

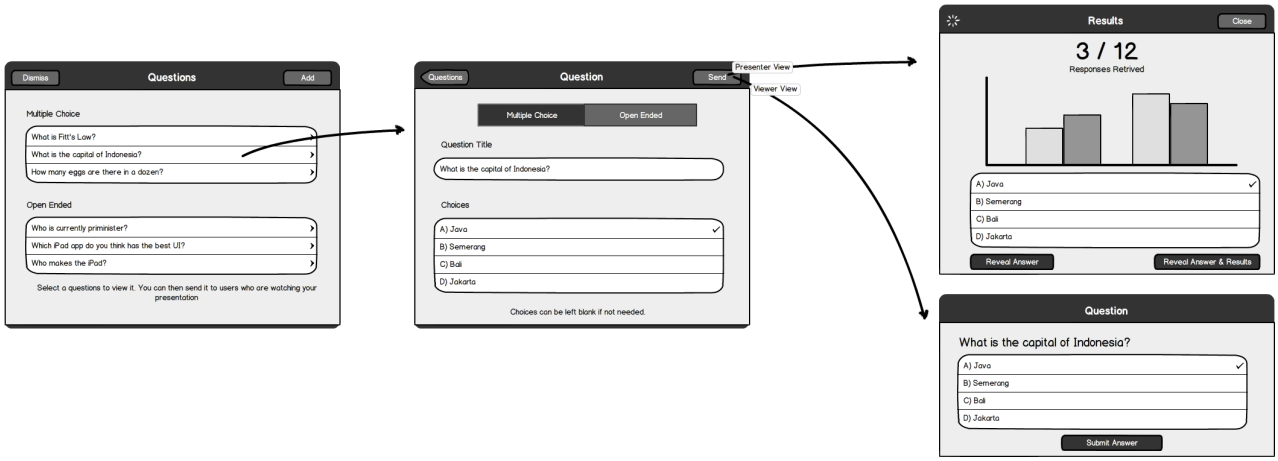


Figure 3.11: Question screens for presenter and viewers.

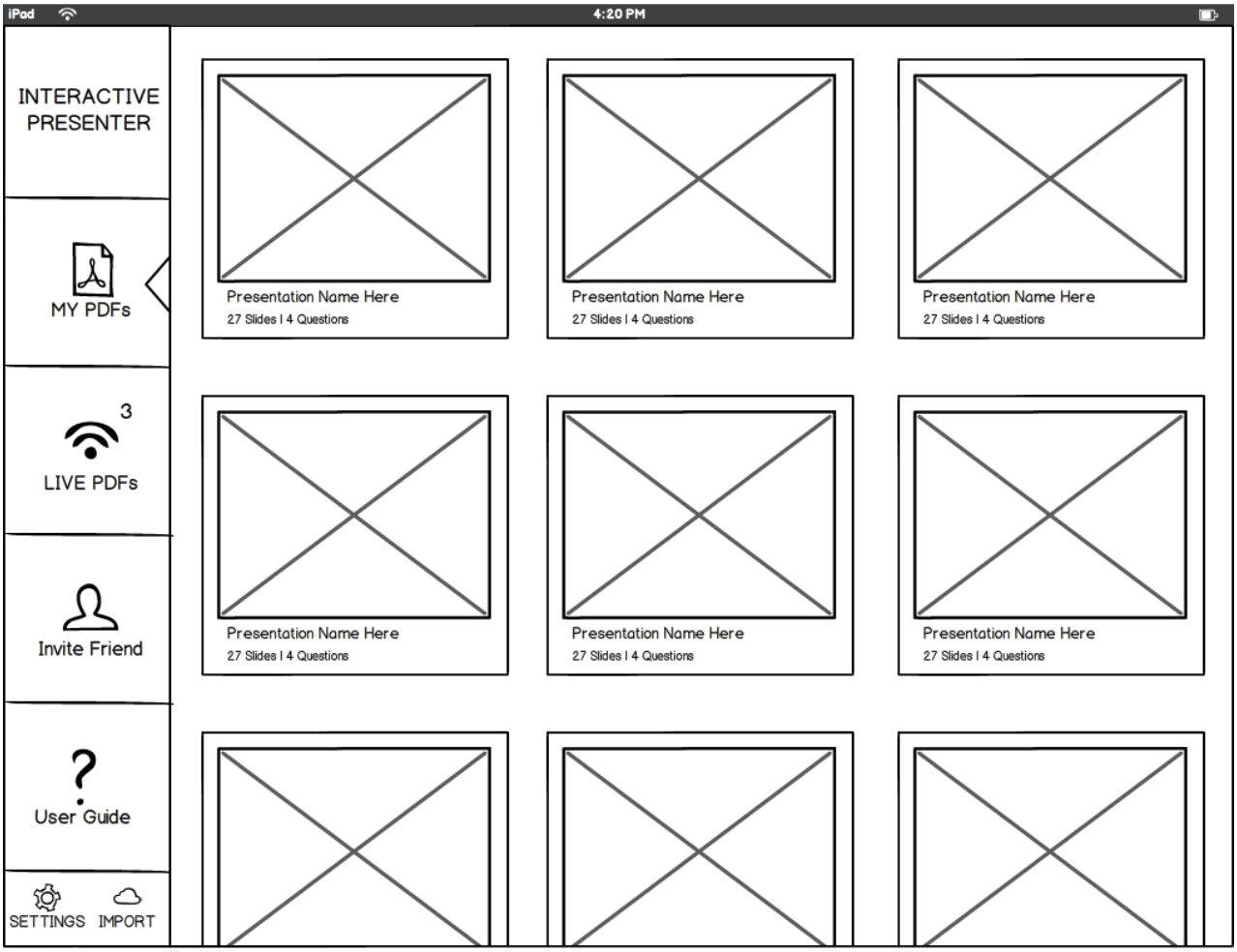


Figure 3.13a: Landscape

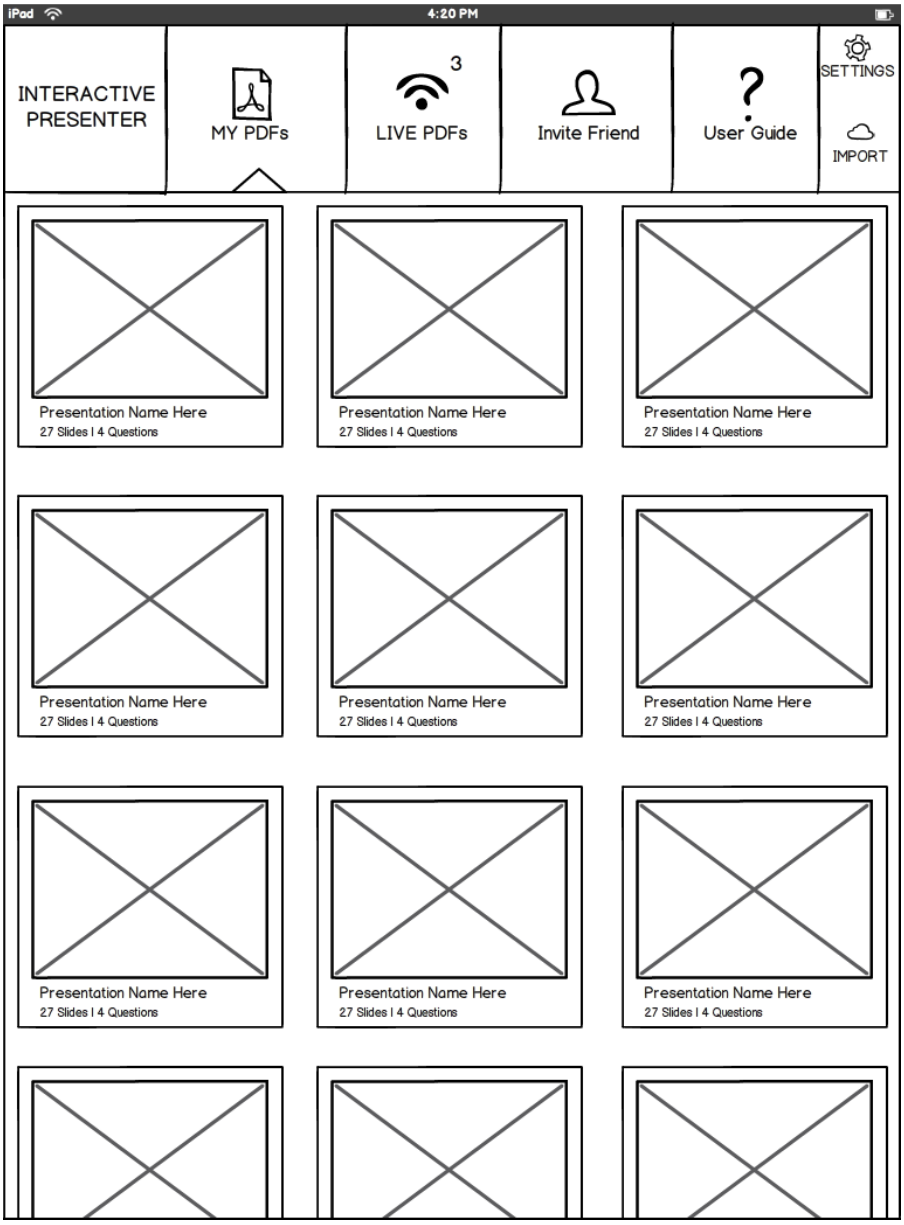


Figure 3.13b: Portrait

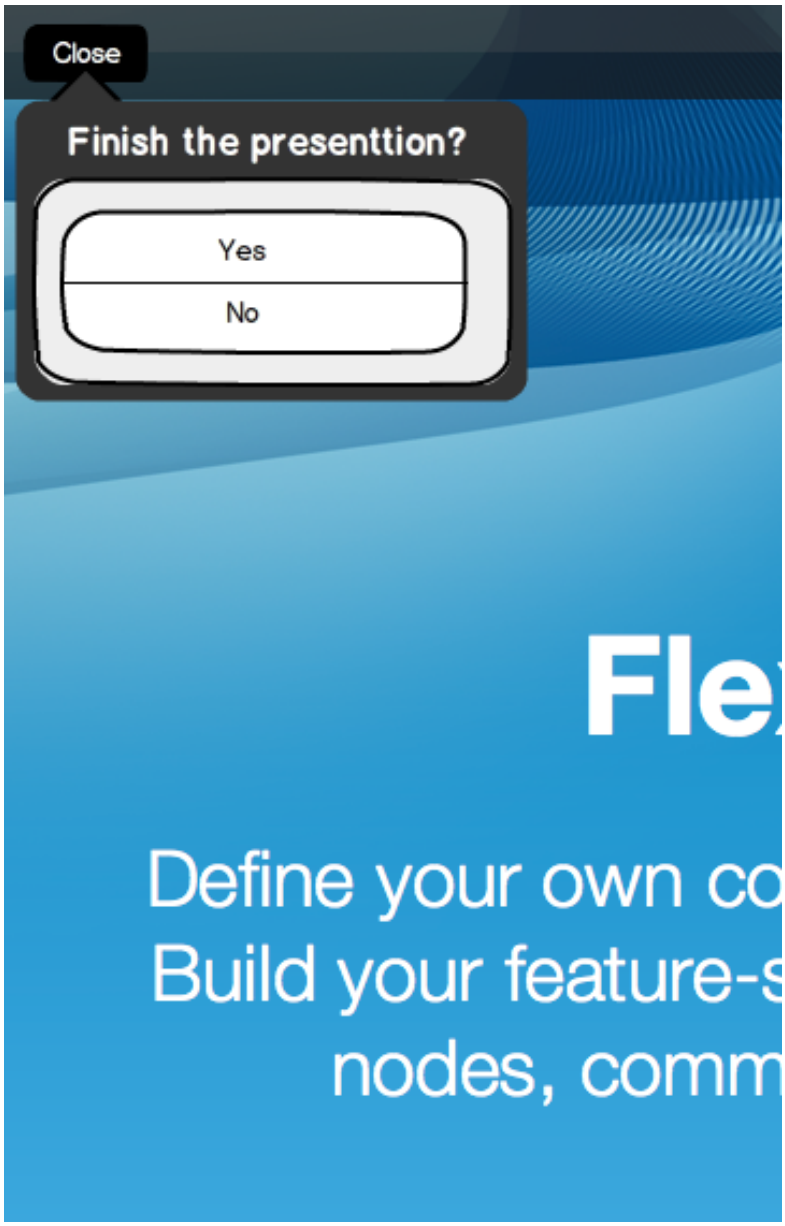


Figure 3.12c: Close prompt.

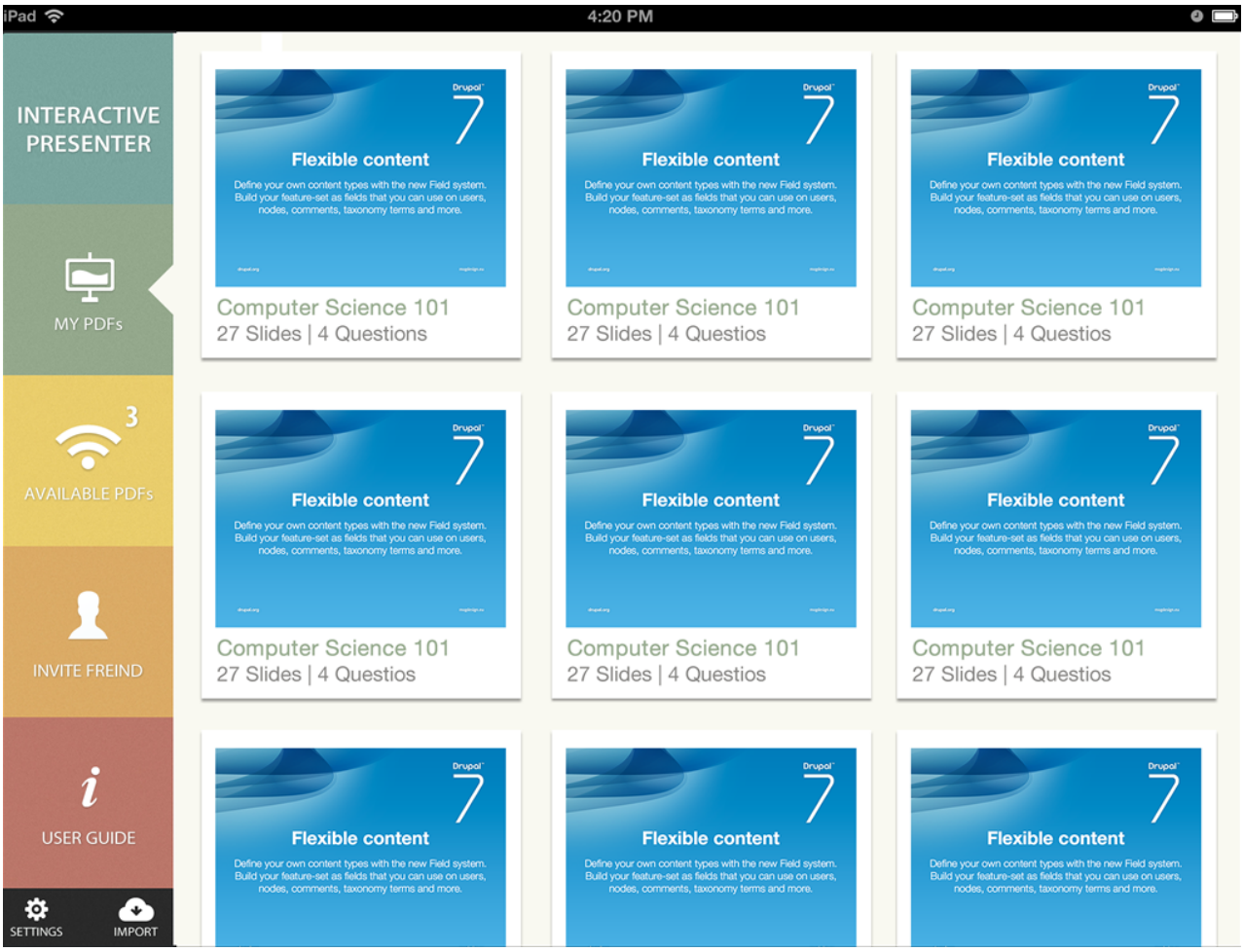


Figure 3.13a: Final design including graphics

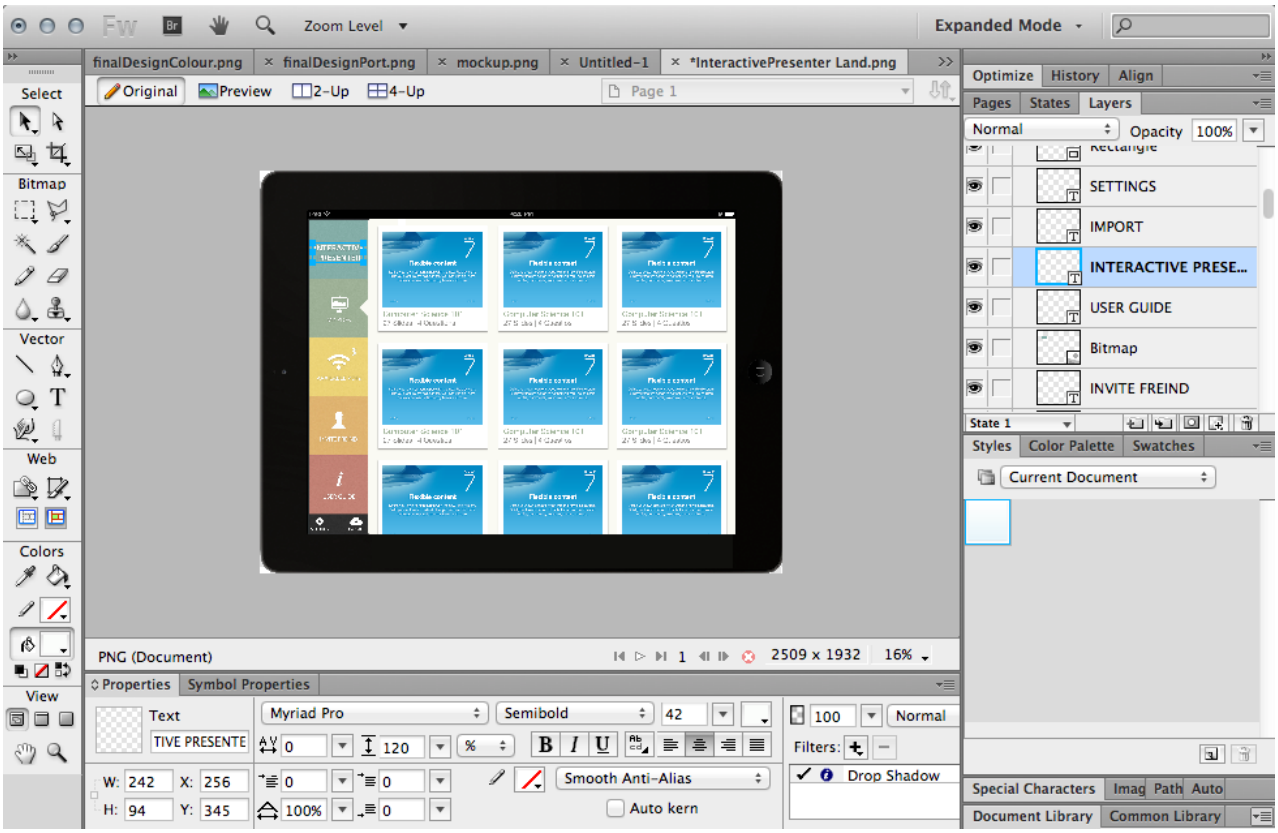


Figure 3.13b: Vectorised design created in Fireworks

Figure 3.14a: Designs using a table view

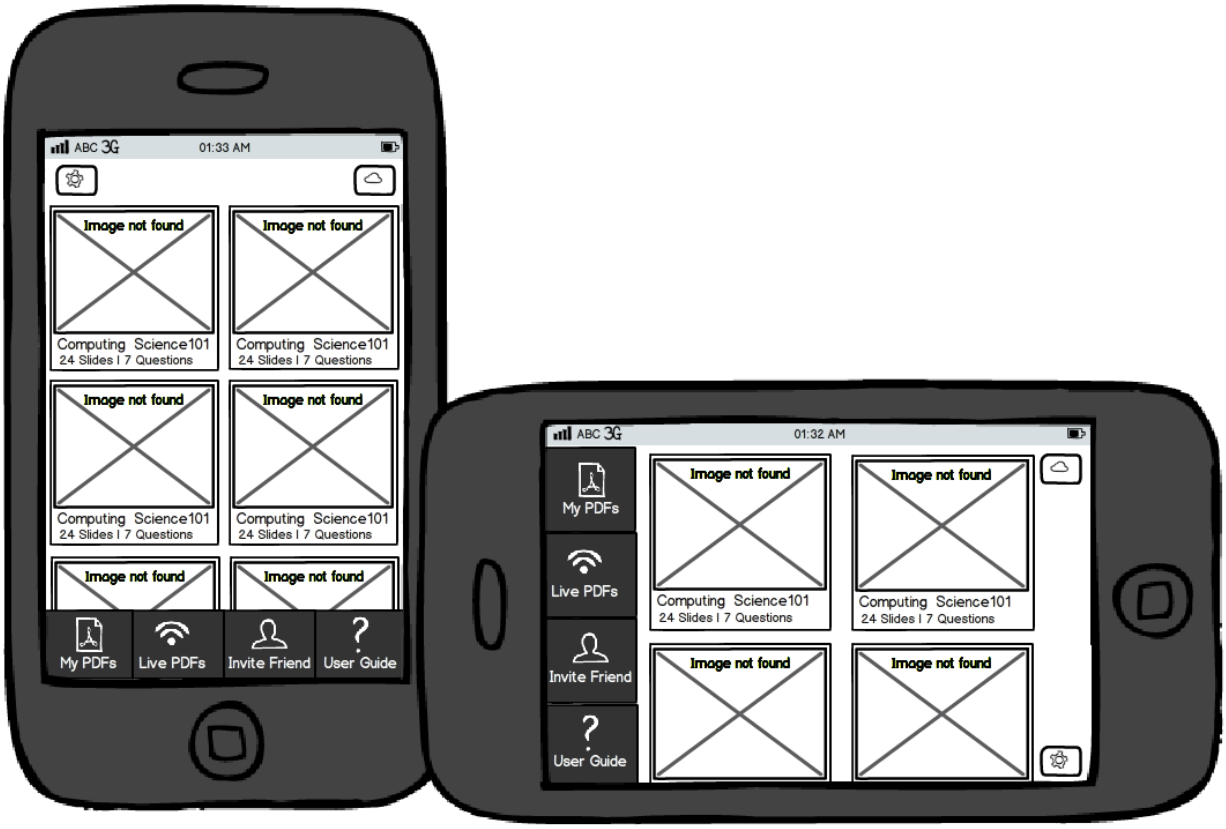


Figure 3.14b: Design with iPad style menu

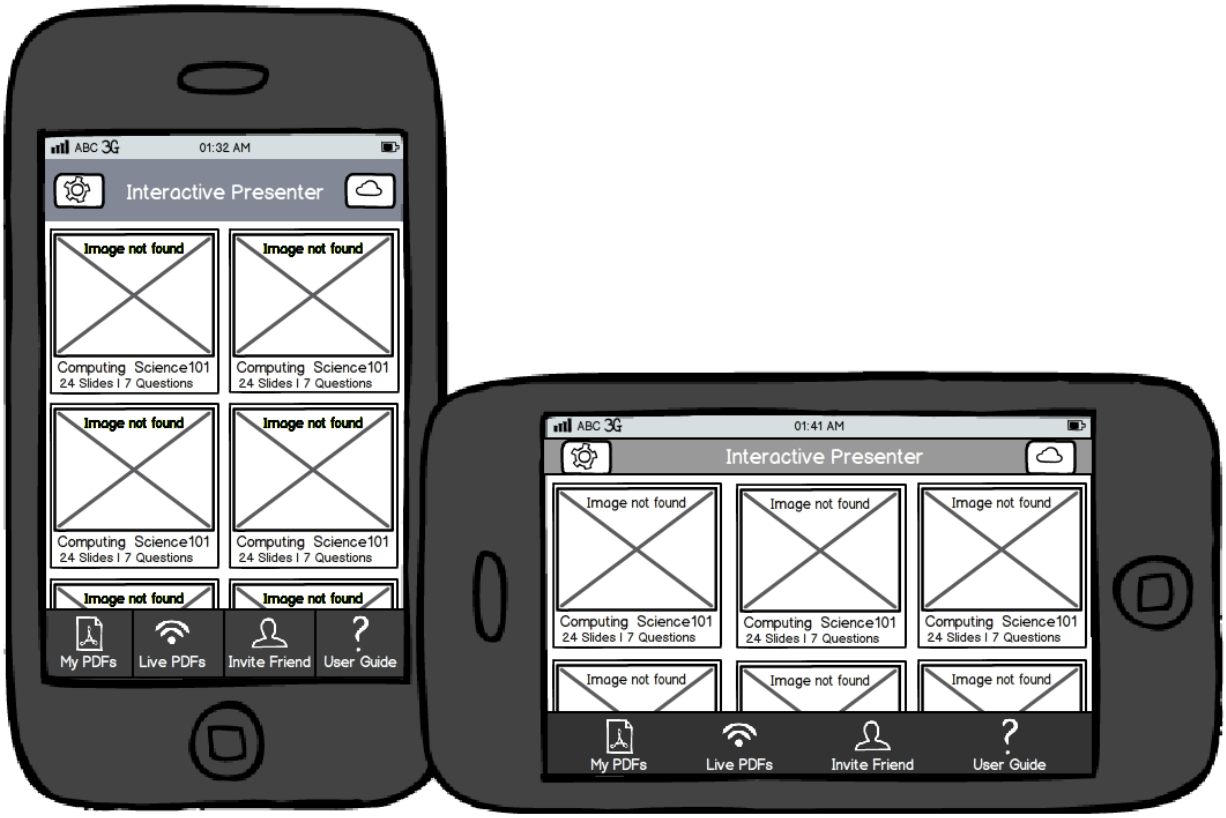


Figure 3.14c: Final design standard tabbar

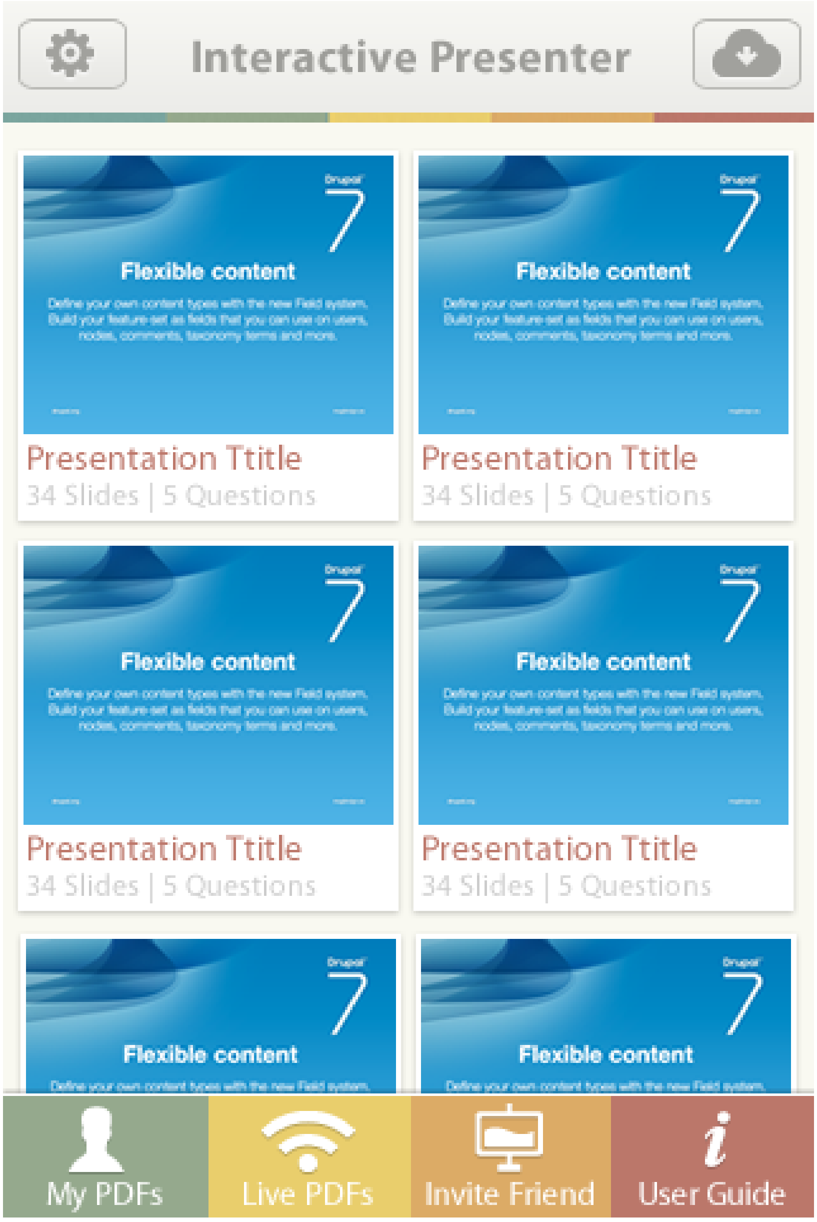


Figure 3.15a: iPad styled tabbar

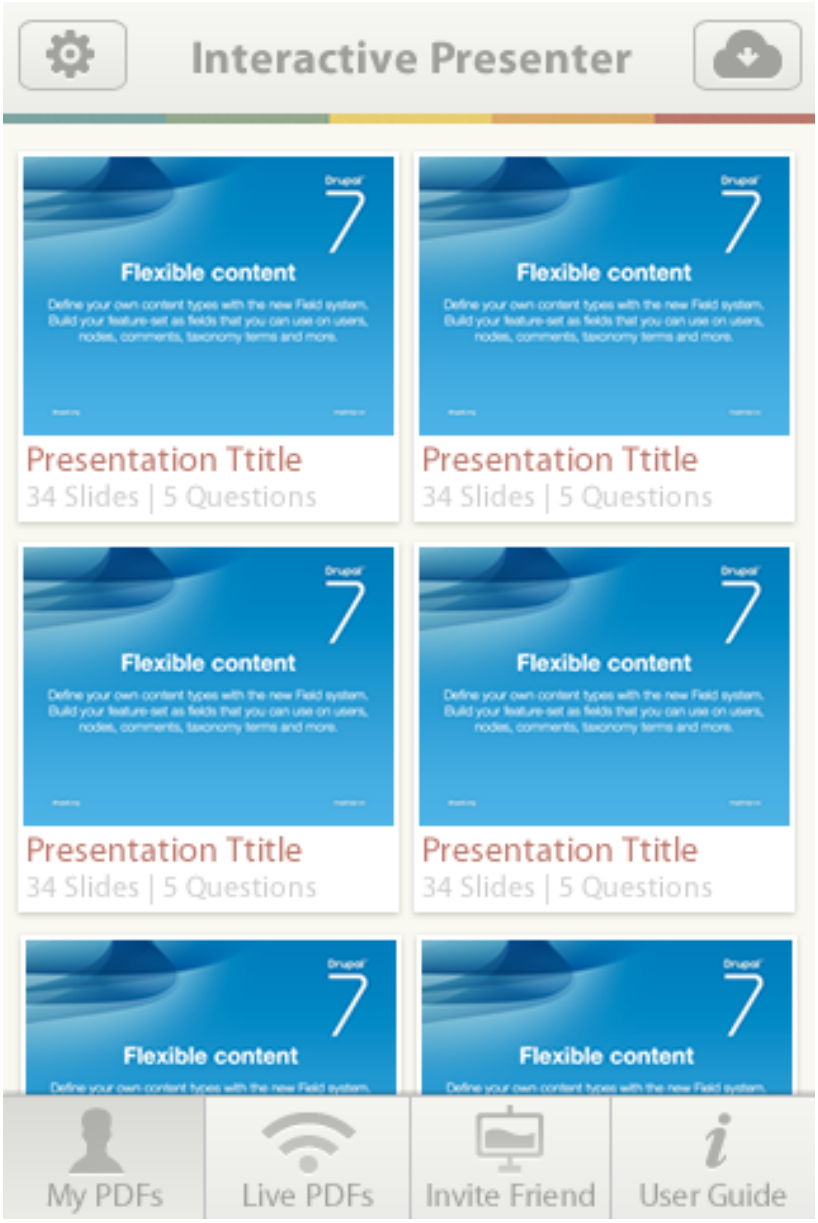


Figure 3.15b: Plain tabbar

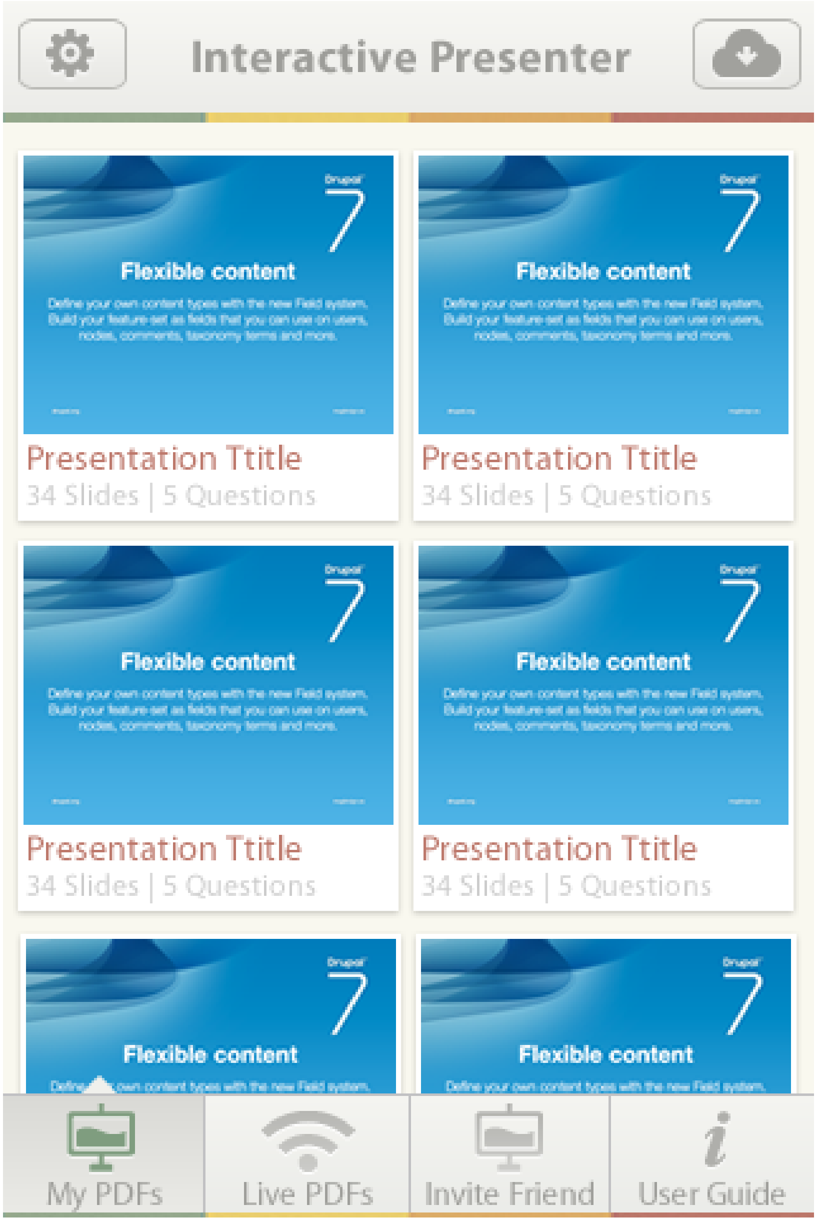


Figure 3.15c: Final design

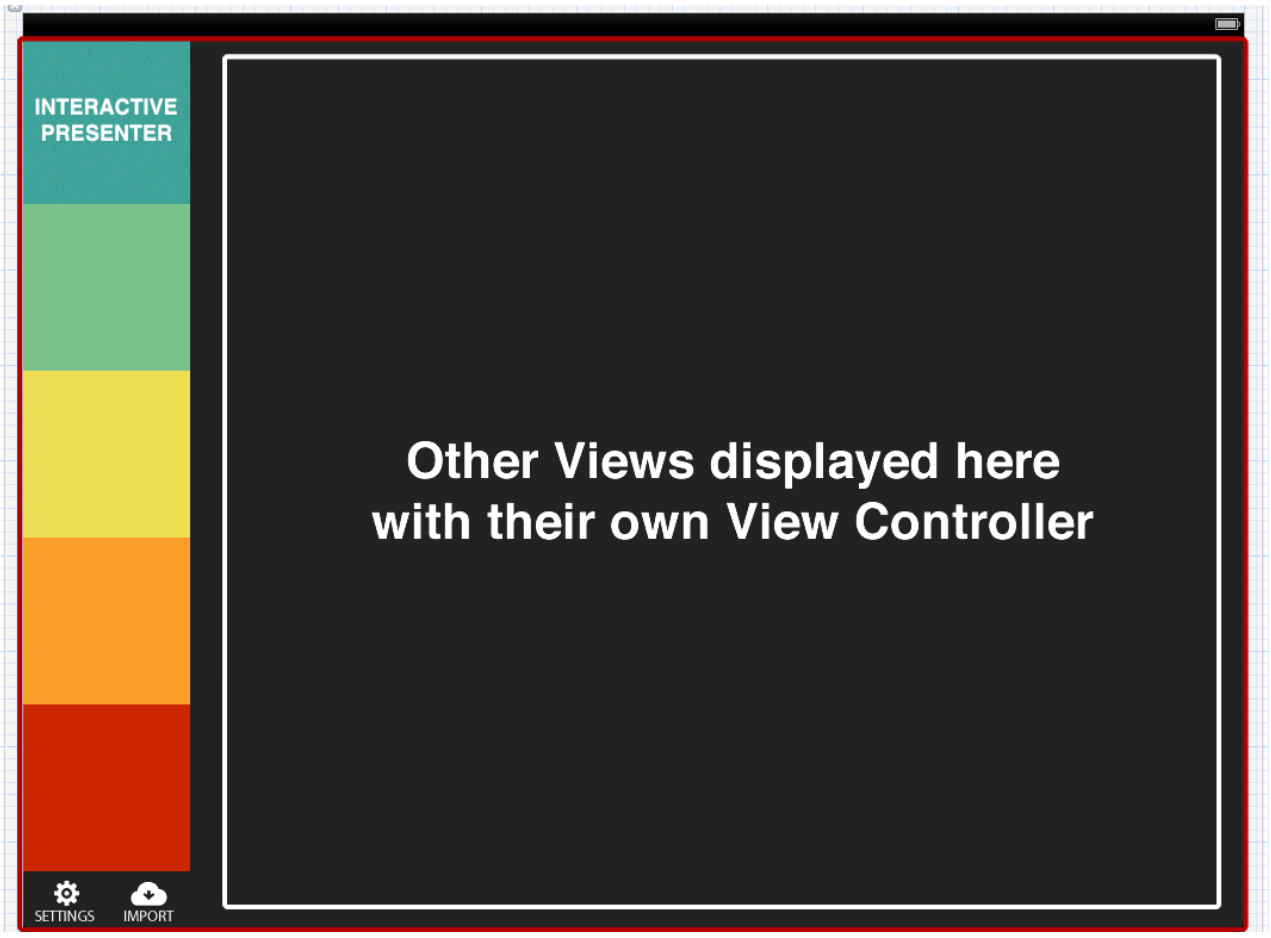


Figure 4.1a: Custom navigation VC

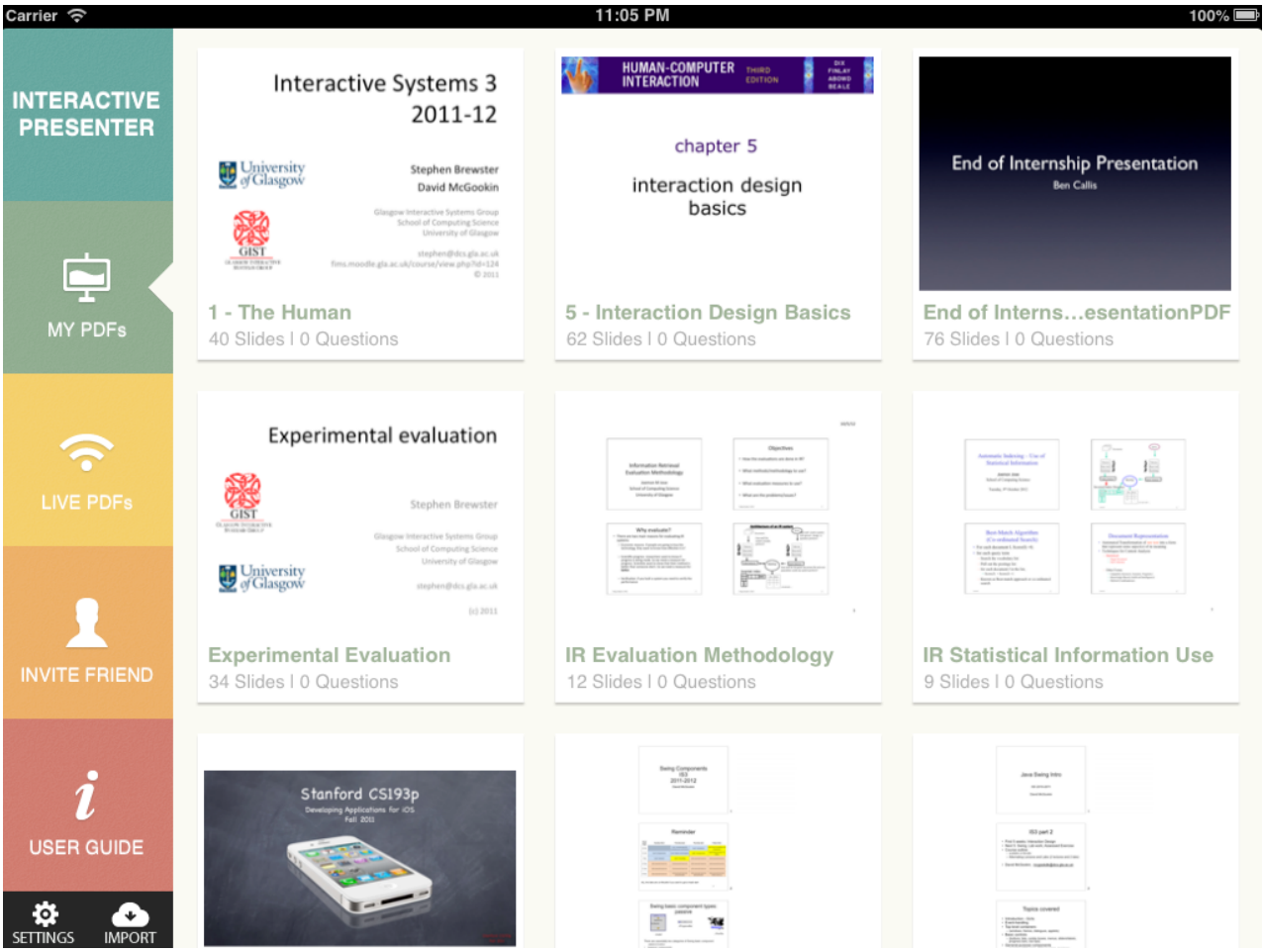


Figure 4.1b: Landscape orientation

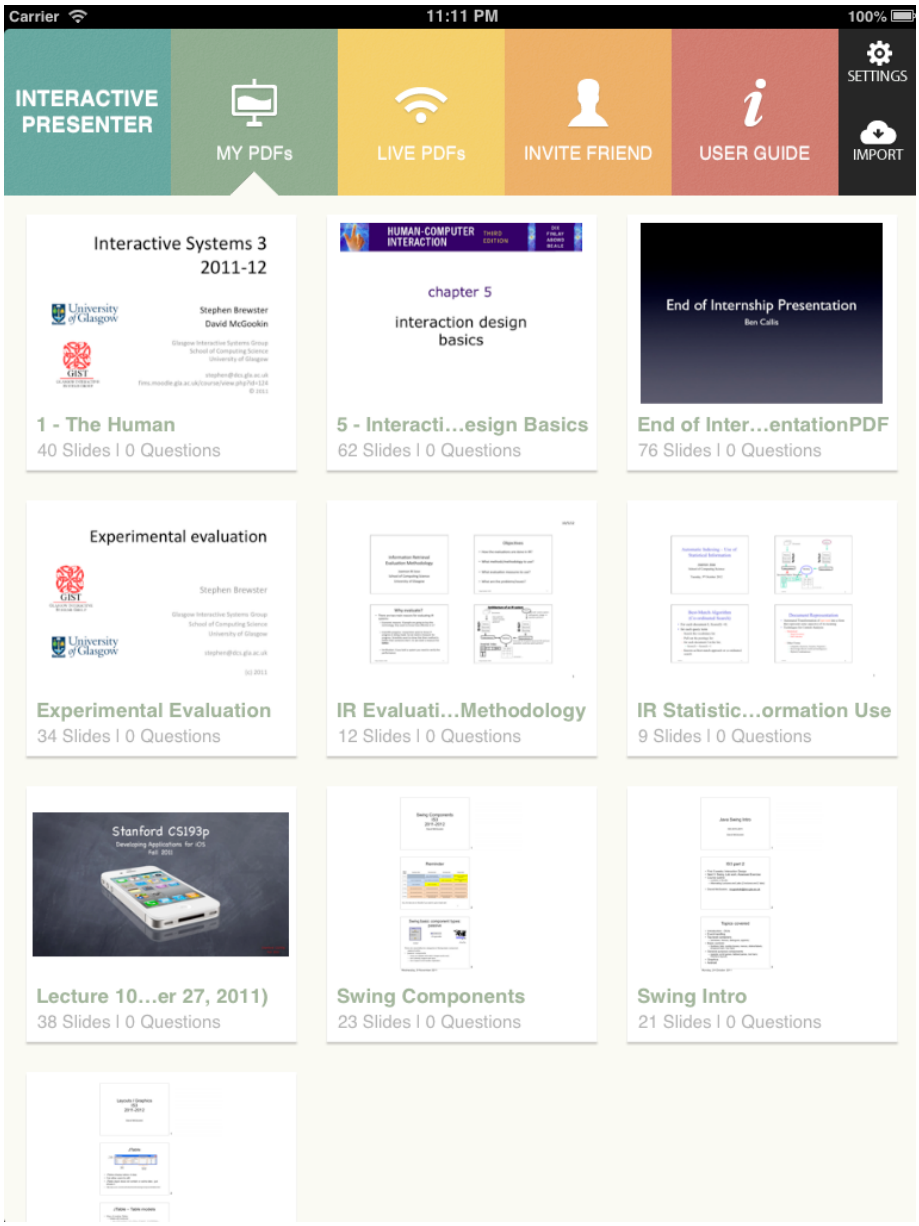


Figure 4.1c: Portrait orientation

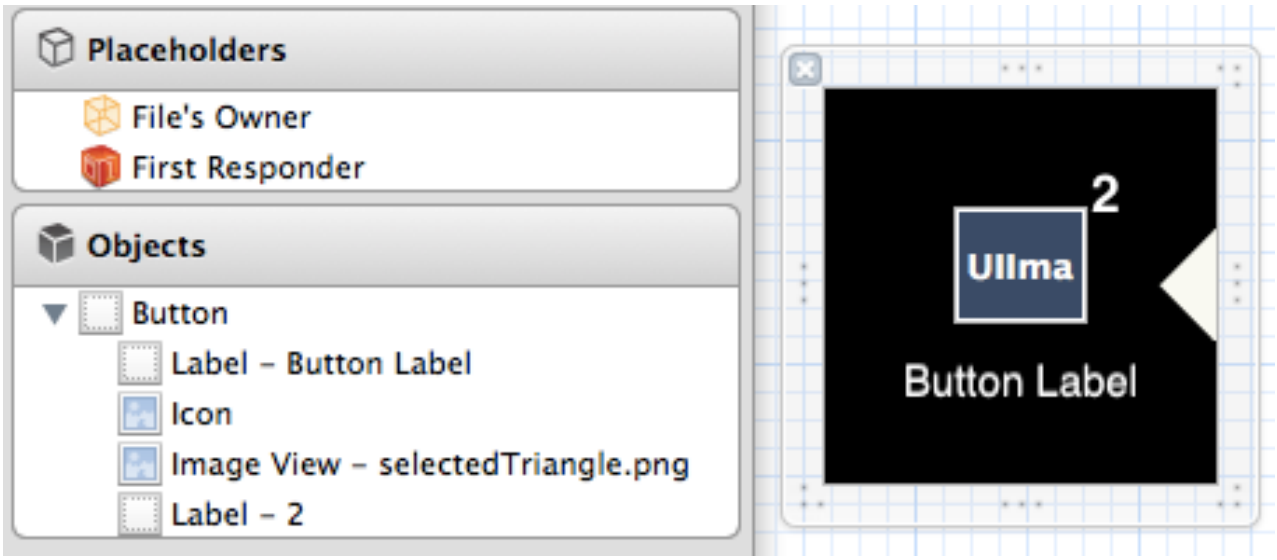


Figure 4.2a: Menu Button View

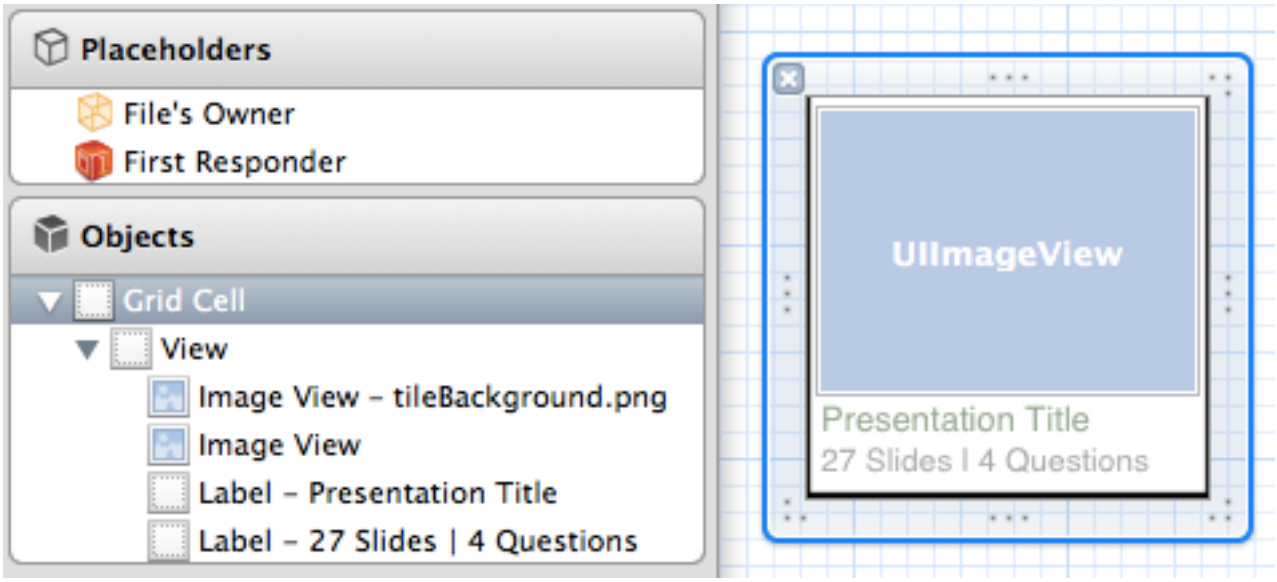


Figure 4.2b: Grid Cell View

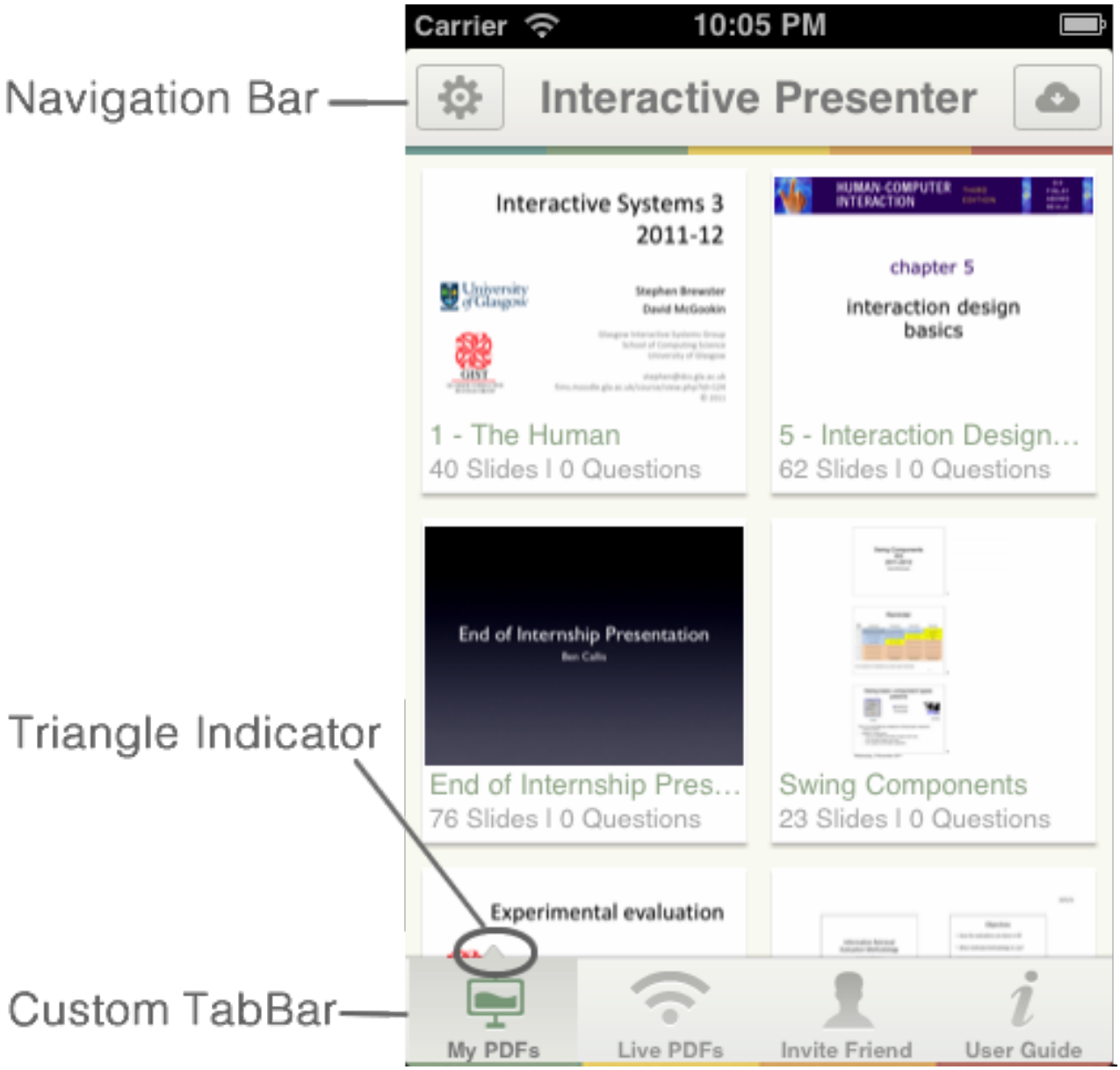


Figure 4.3a: iPhone Landscape

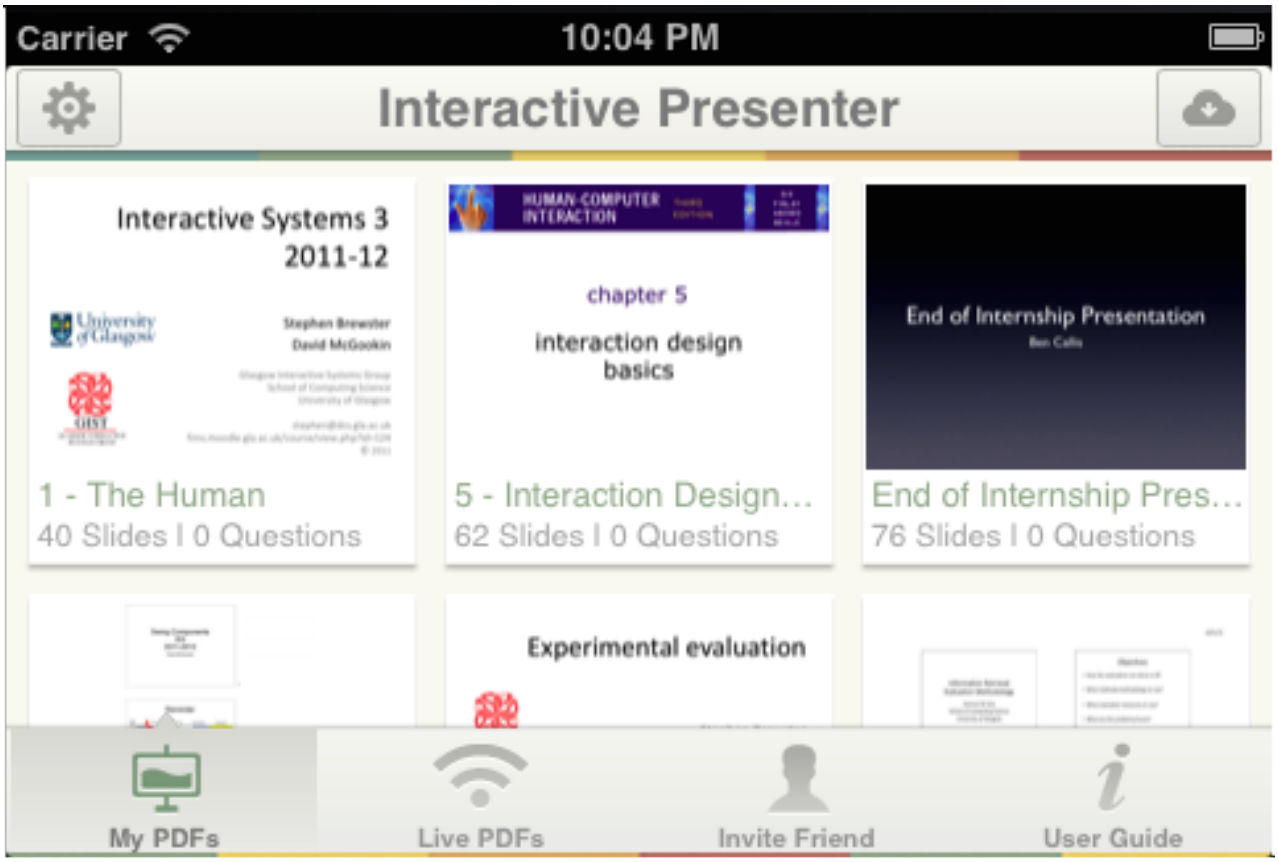


Figure 4.3b: iPhone Portrait

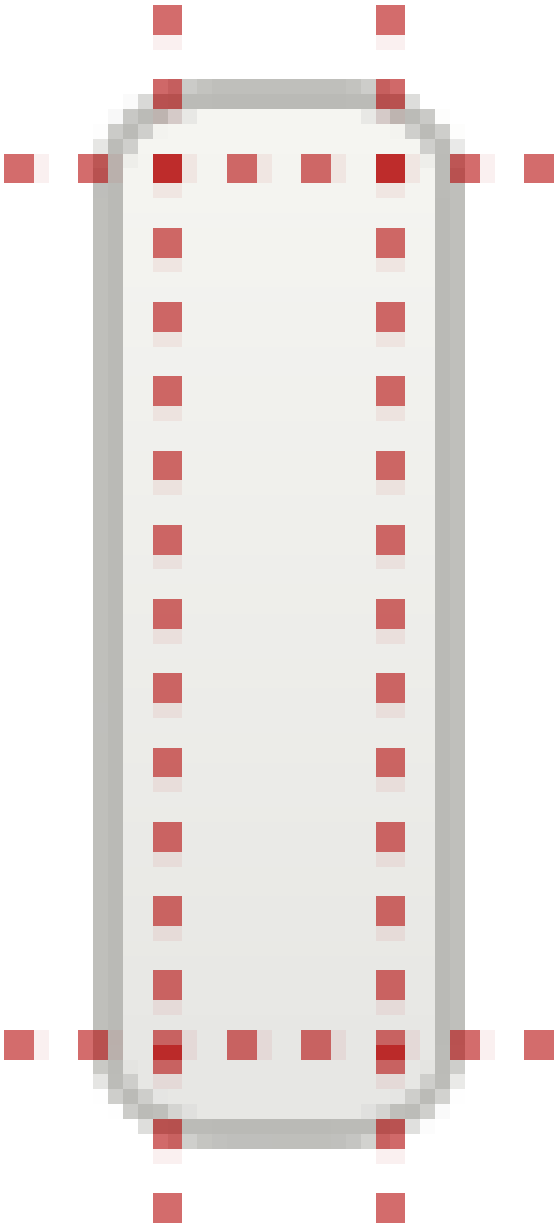


Figure 4.4a: Scaleable button image

Figure 4.4b: Stretched button

Figure 4.4c: Scaled button

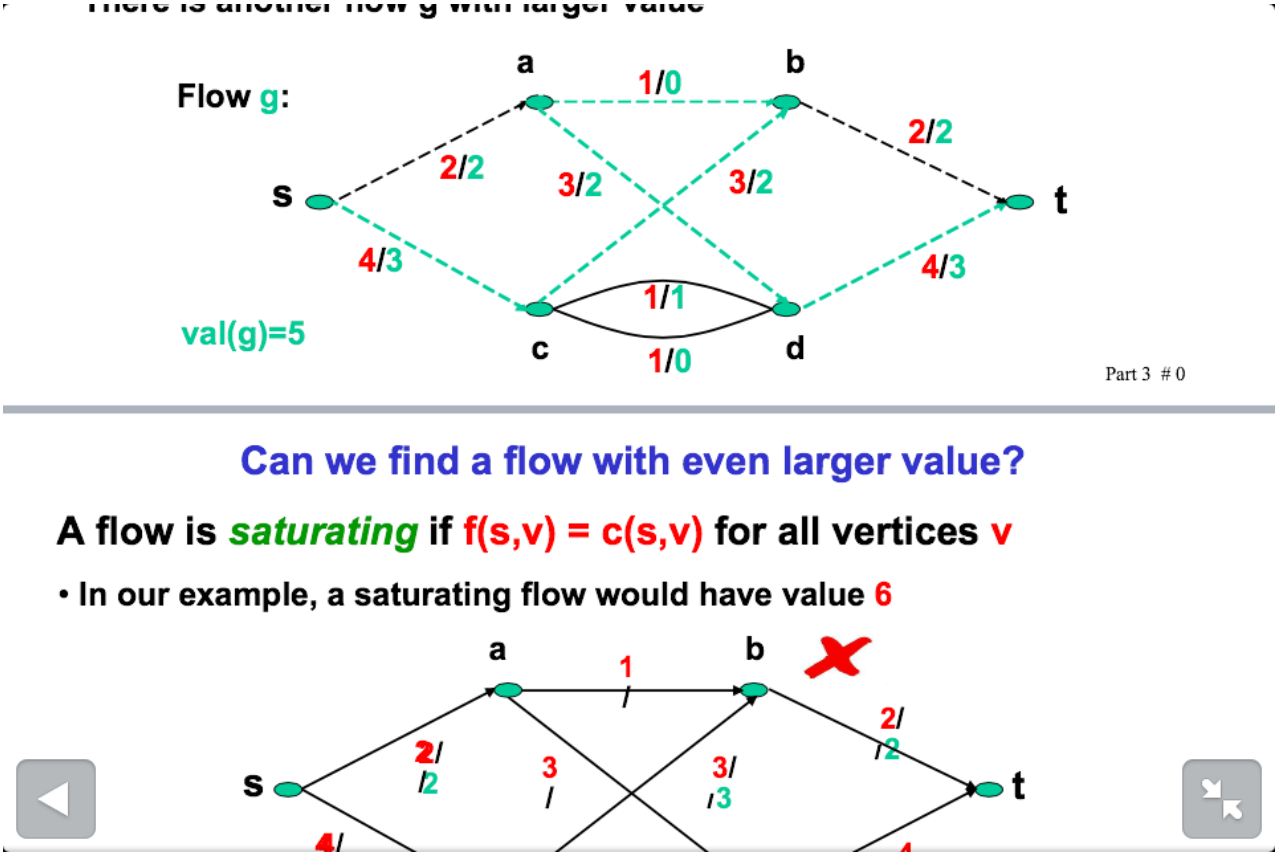


Figure 4.5: UIWebView navigation

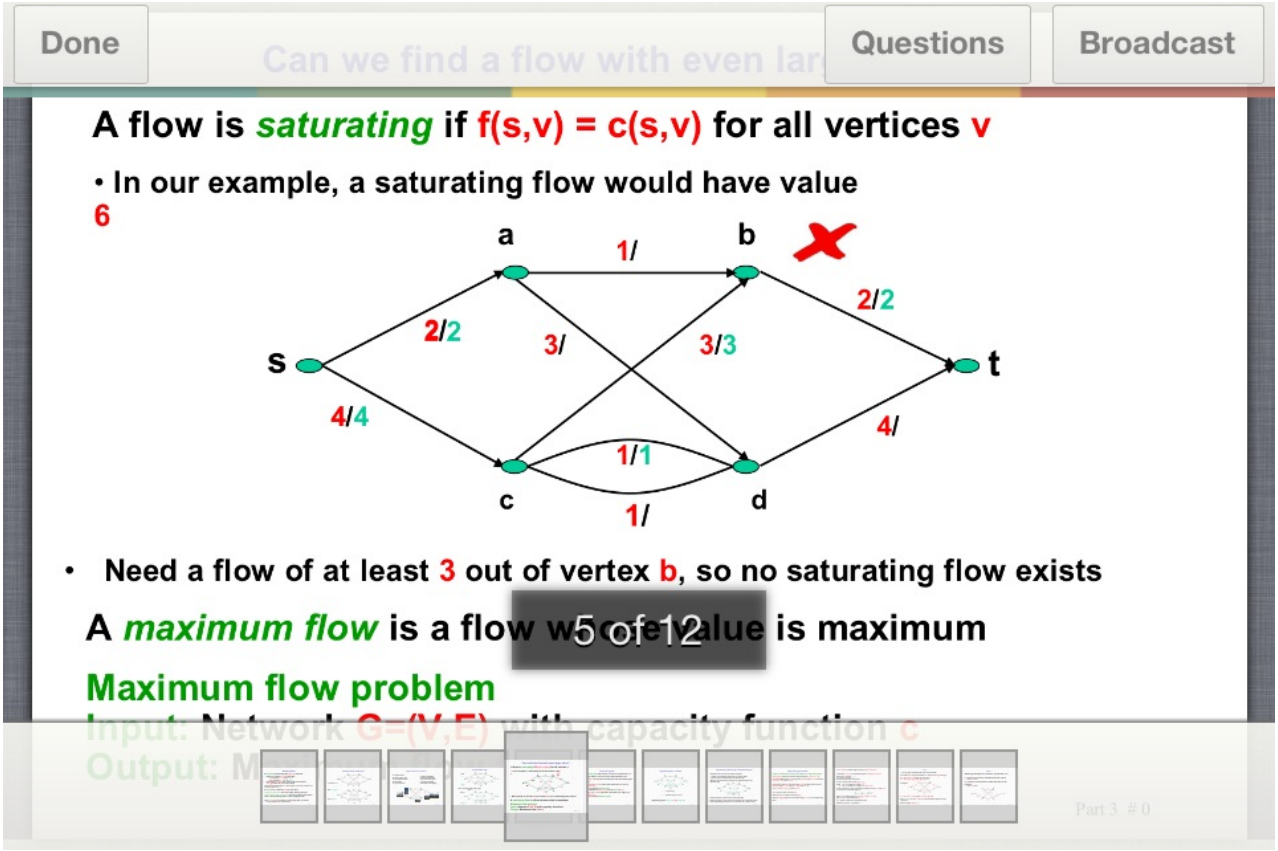


Figure 4.6: Customised VFR Reader

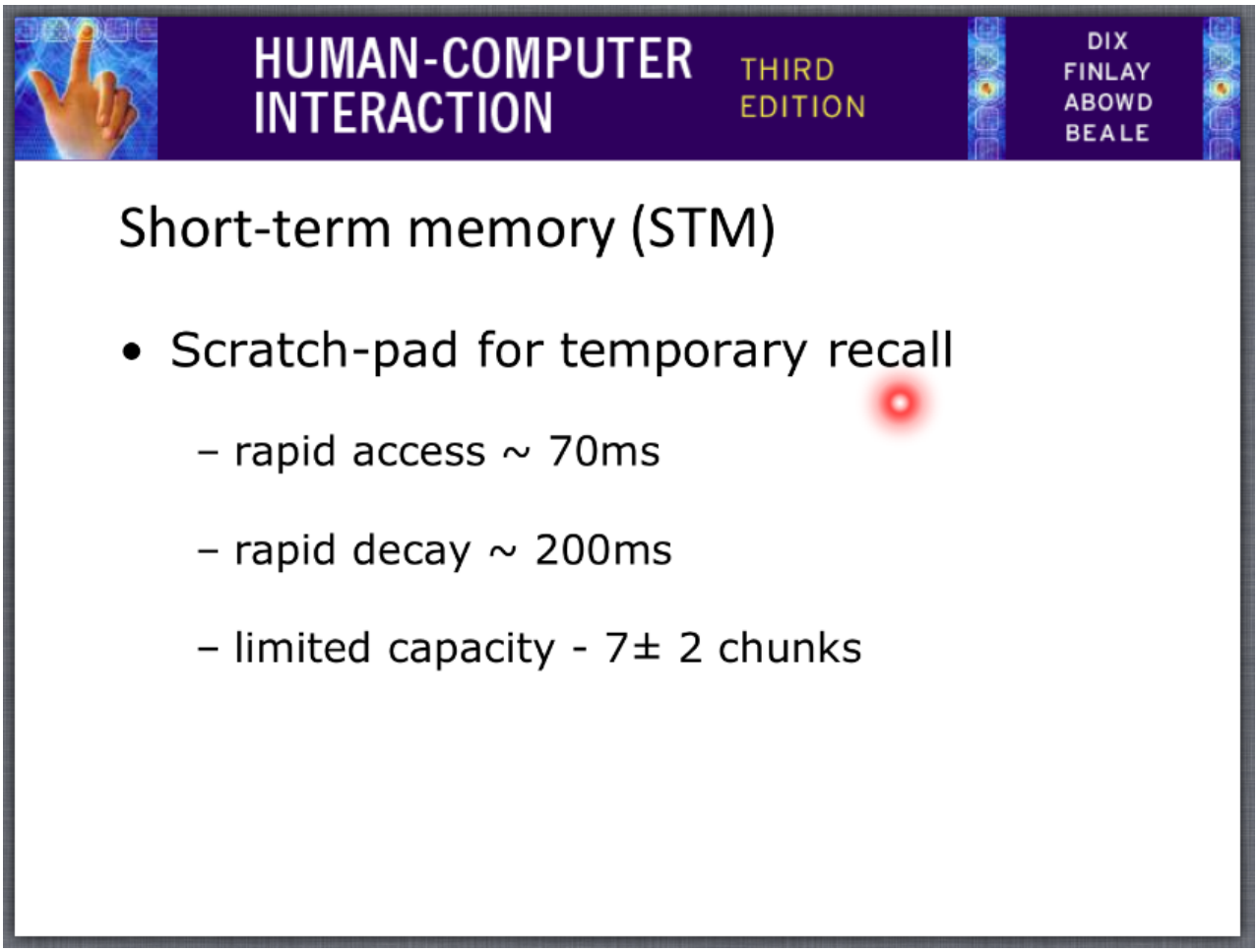


Figure 4.7: Laser pointer

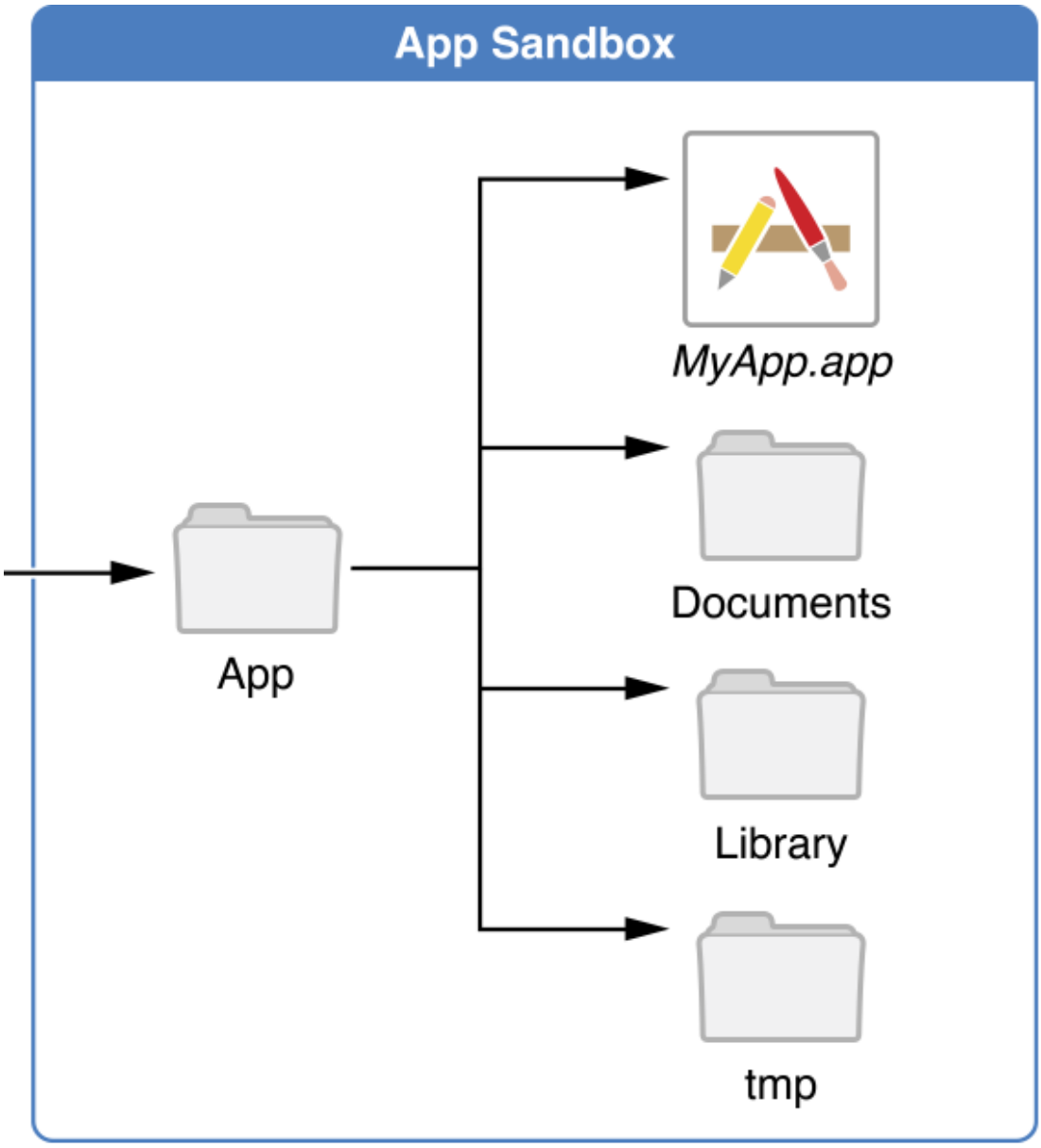


Figure 4.8: iOS app sandbox

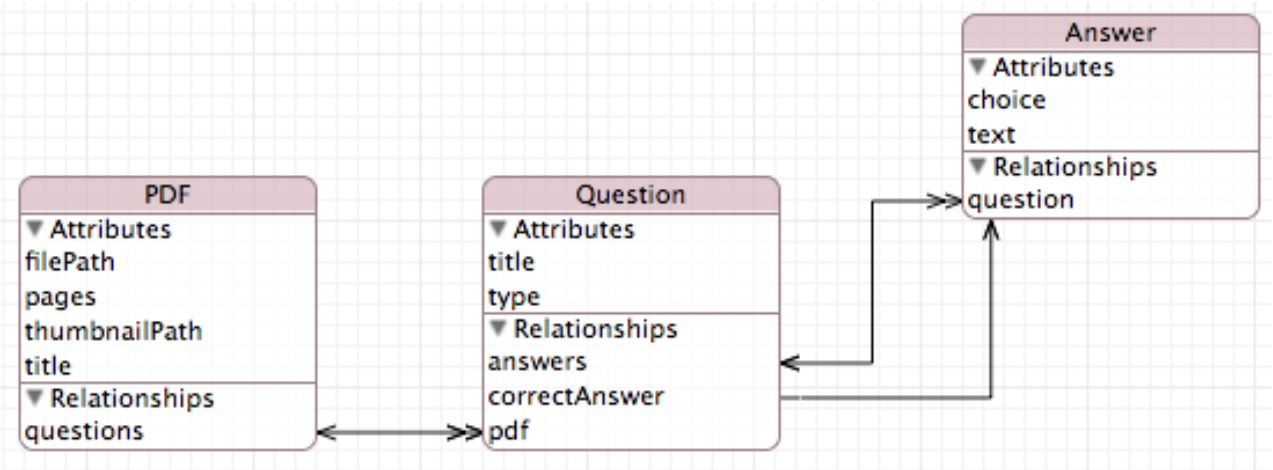


Figure 4.9: Database model.

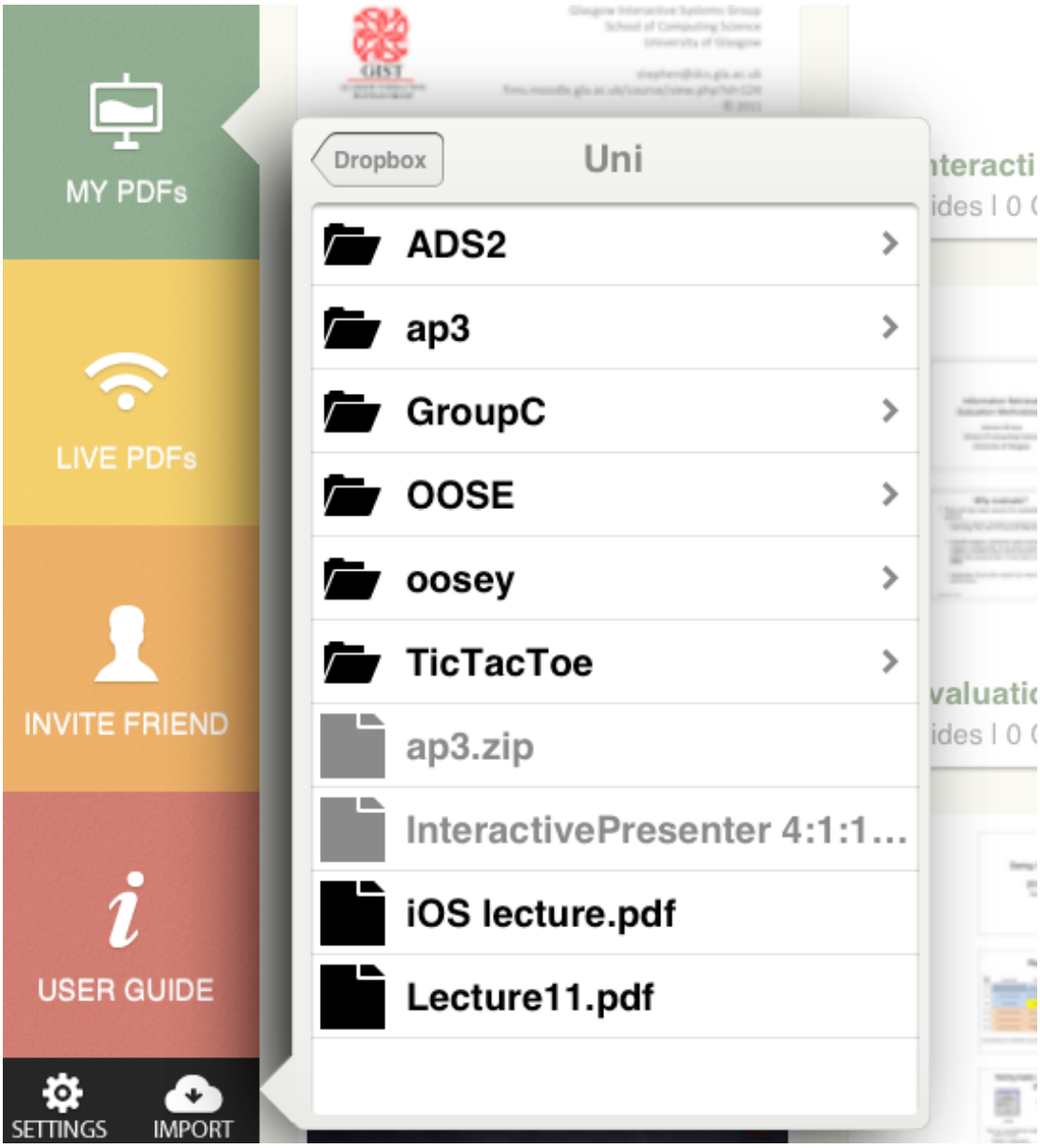


Figure 4.10a: FilePicker presented in a popover on iPad

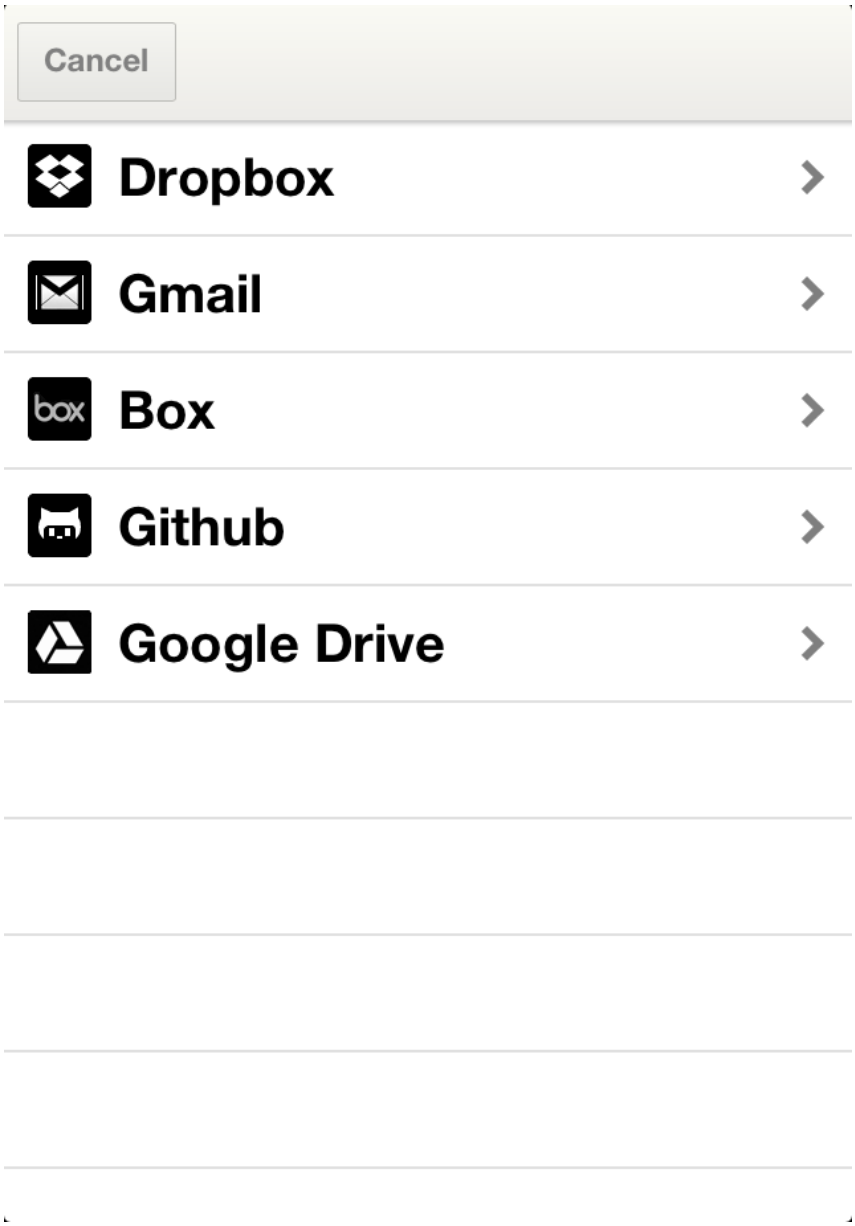


Figure 4.10b: FilePicker presented modally on iPhone

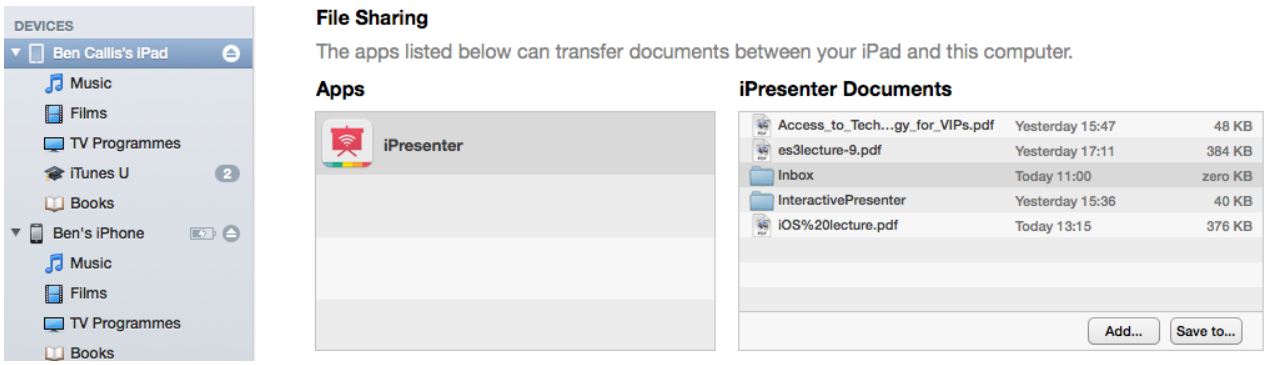


Figure 4.11: iTunes file sharing

Figure 4.12: Standard Game Kit picker

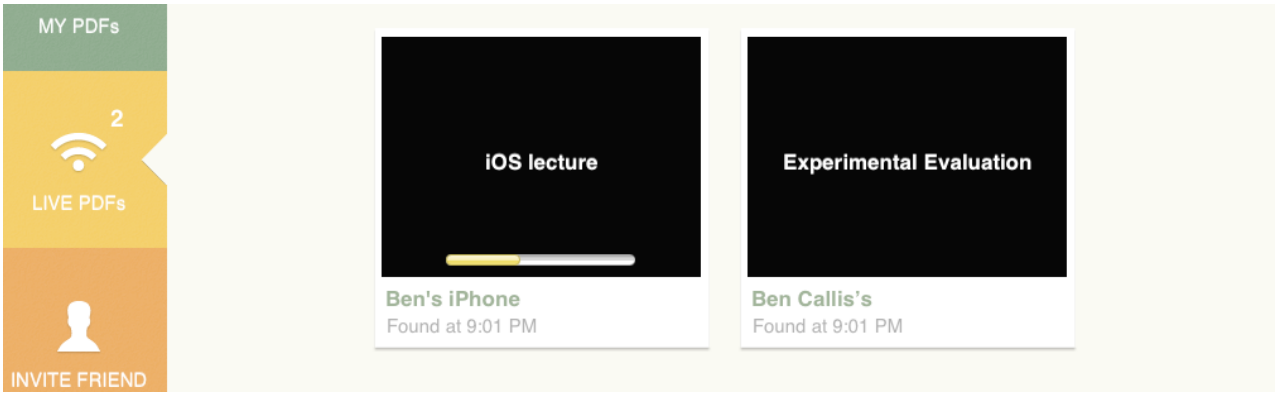


Figure 4.13: Live PDFs VC - Custom picker iPad

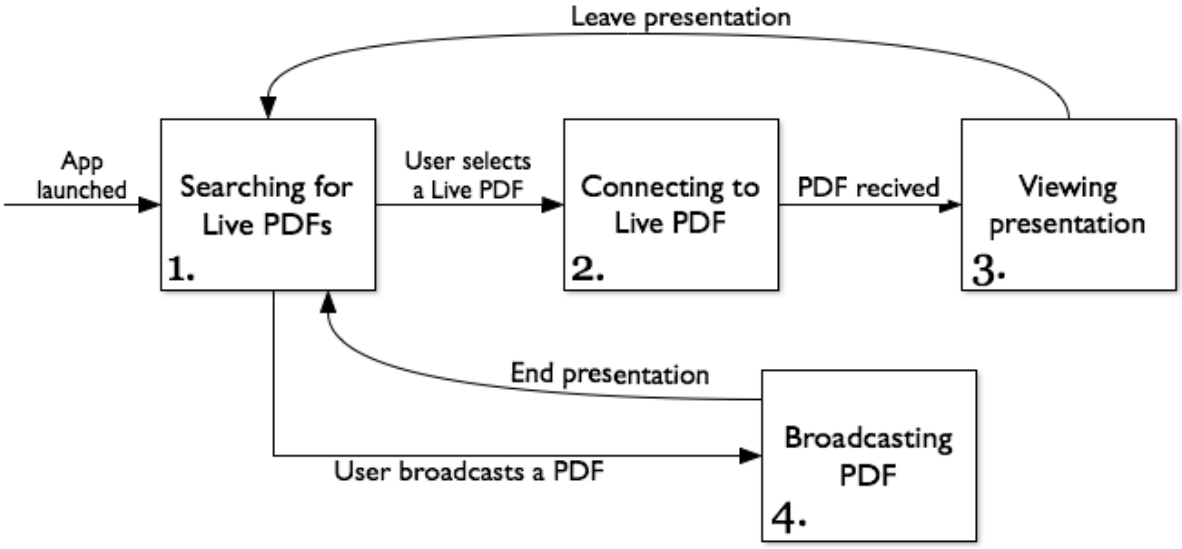


Figure 4.14: Networking States

UniqueID	PDF MD5	PDF Name	Payload ...	PDF Size	Payload Start	Payload Length

Figure 4.15: PDF packet

Figure 4.16a: Presenter View

Figure 4.16b: Viewer View

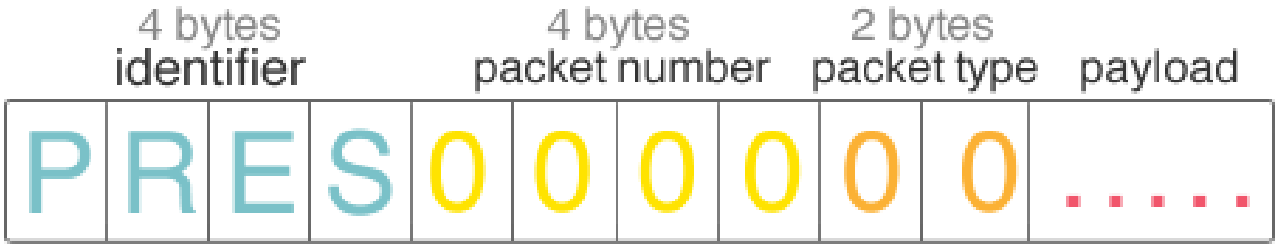


Figure 4.17: 10 byte Packet Header

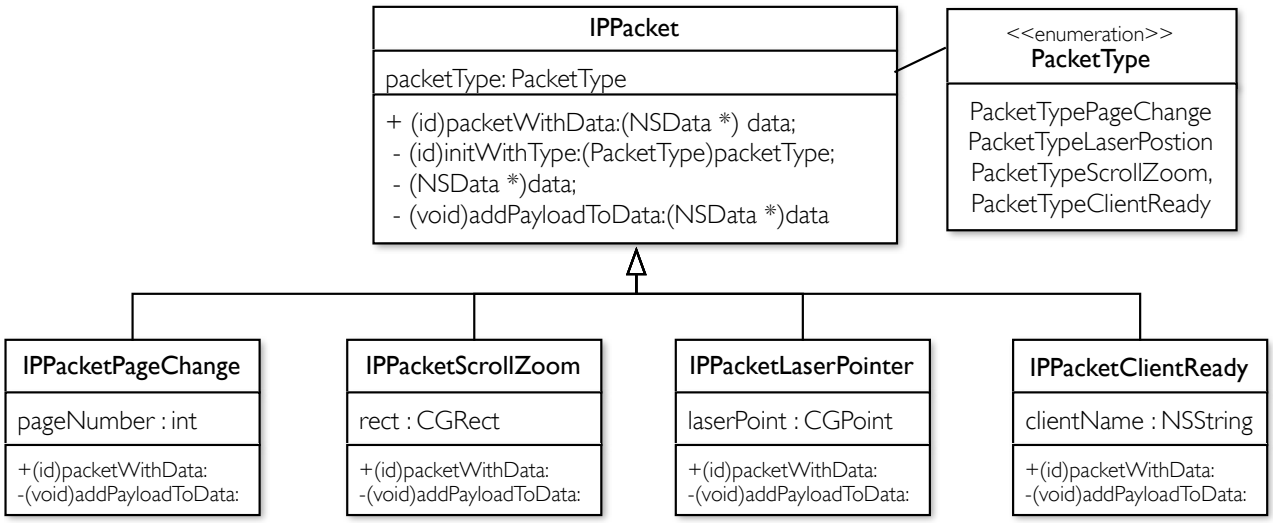


Figure 4.18: Packet classes *(initial 4)*. Using the Factory design pattern.

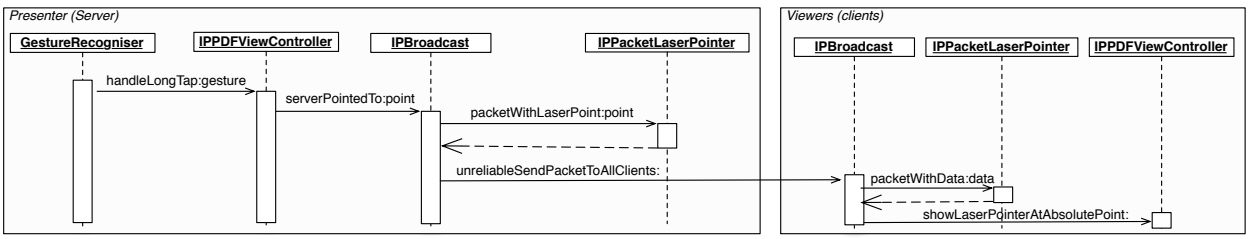


Figure 4.19: Laser pointer packet creation and distribution

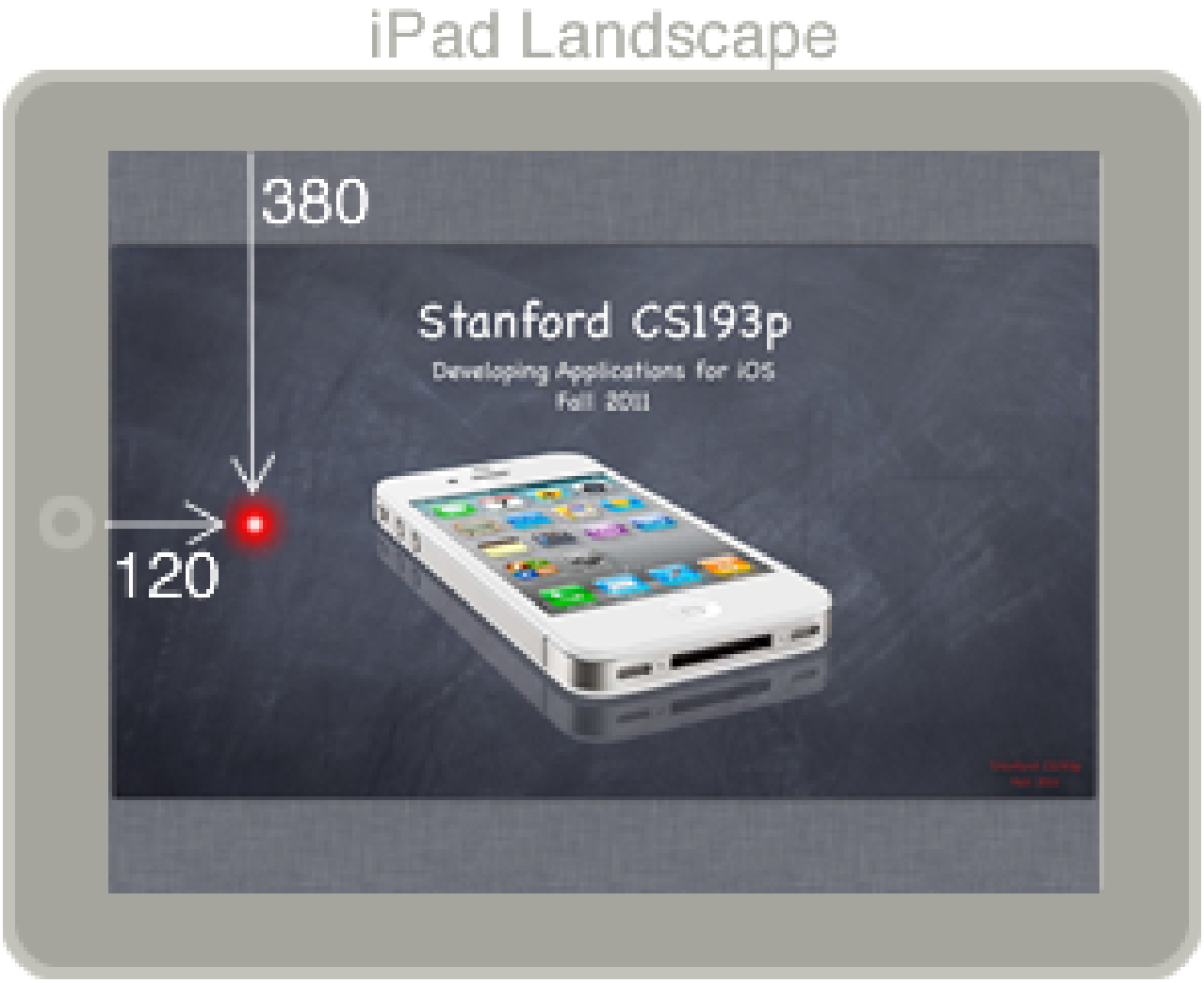


Figure 4.20a: Presenter View

Figure 4.20b: Viewer View

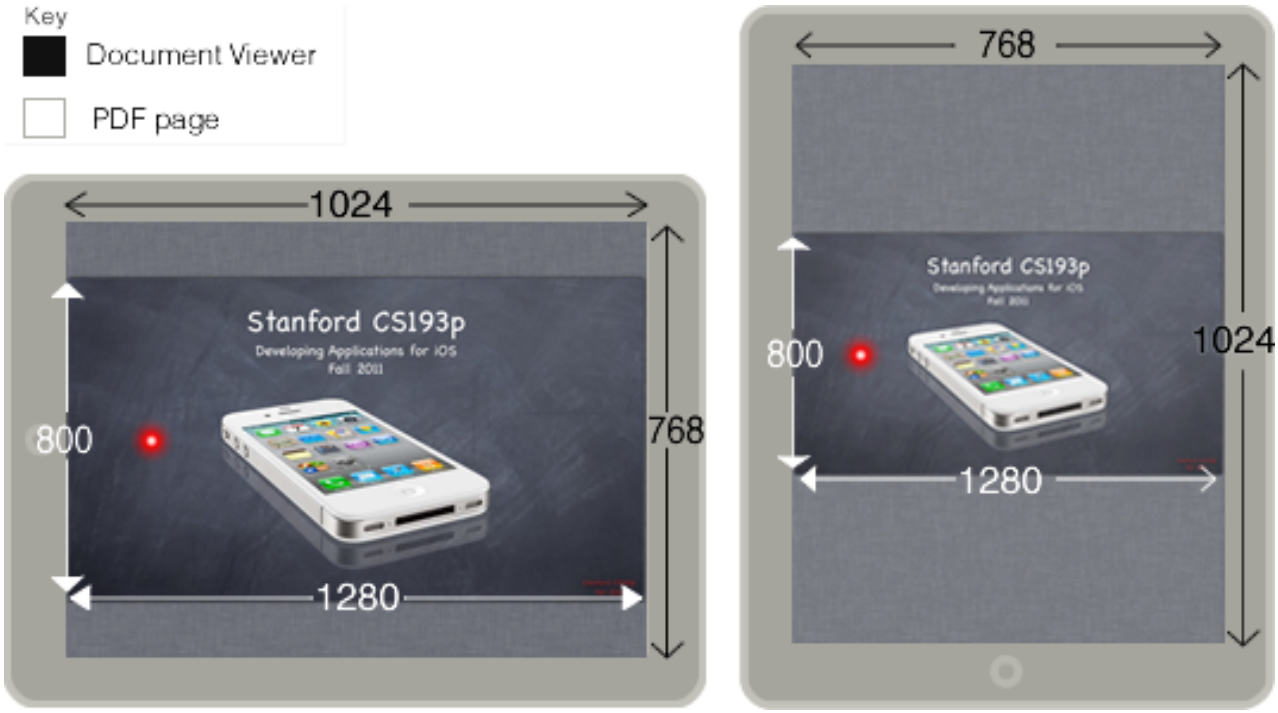


Figure 4.21: View co-ordinate systems

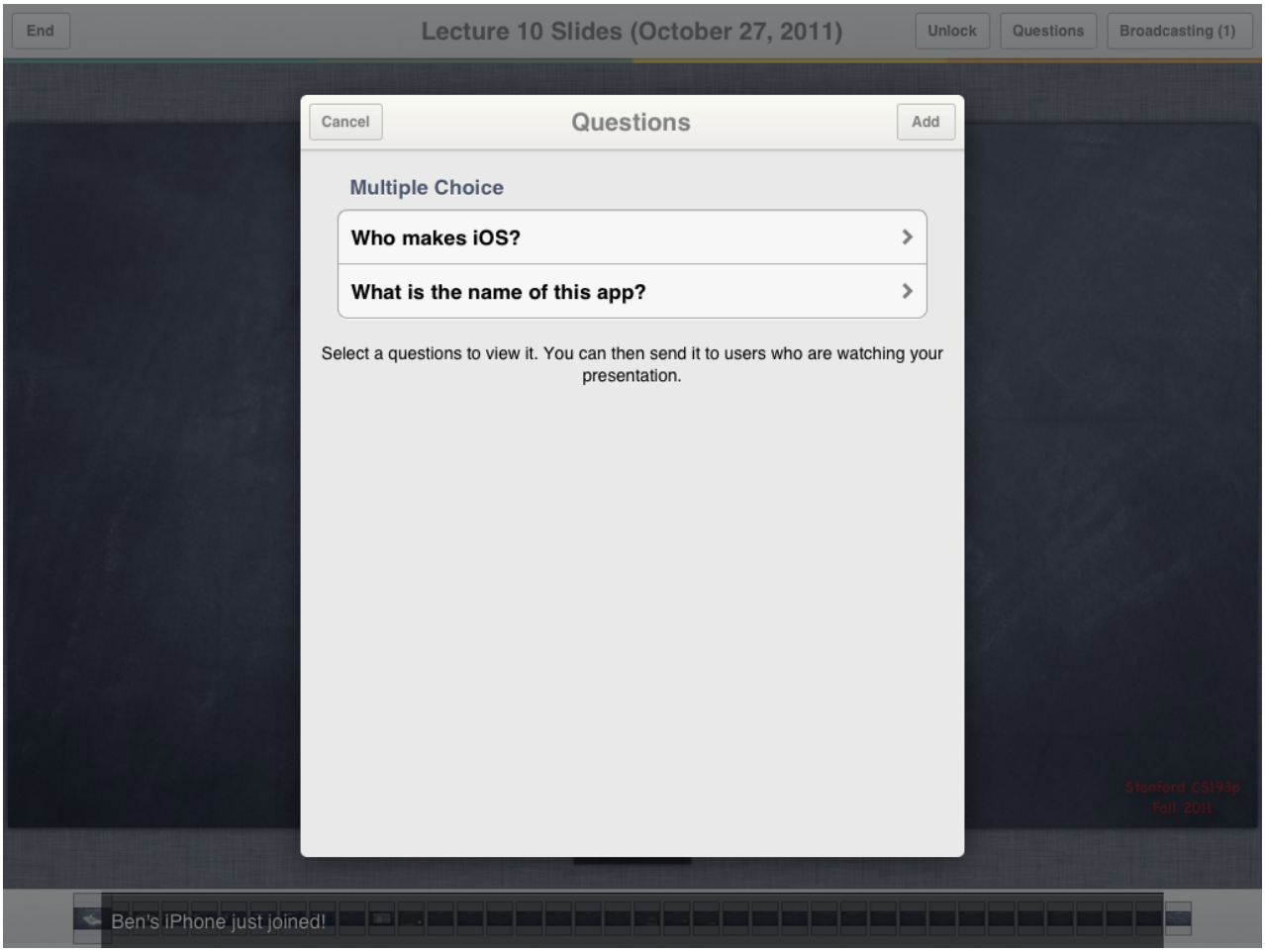


Figure 4.22a: iPad

Figure 4.22b: iPhone

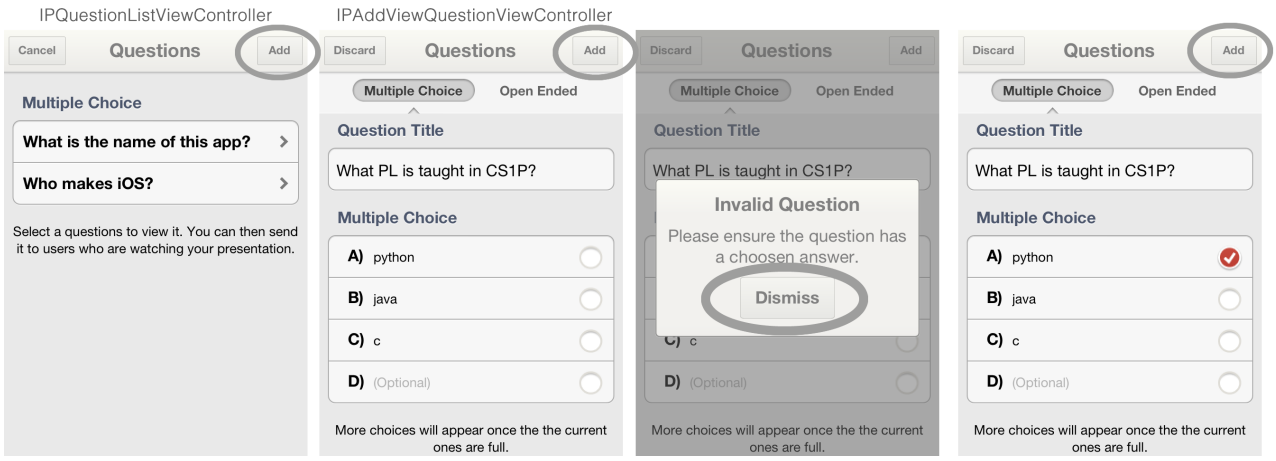


Figure 4.23: Add question

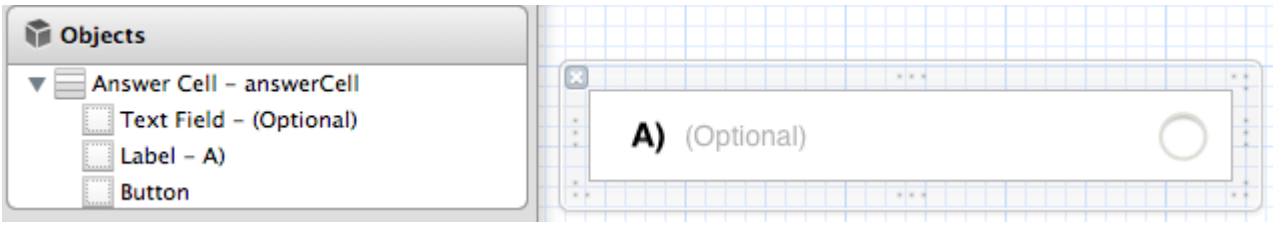


Figure 4.24: Multiple choice custom answer cell

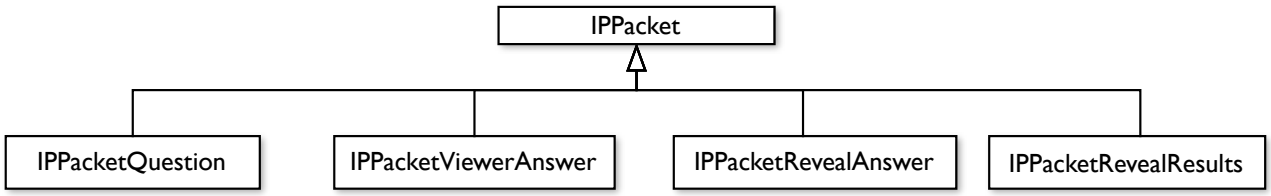


Figure 4.25: Four additional packet types defined for transmitting questions and answers

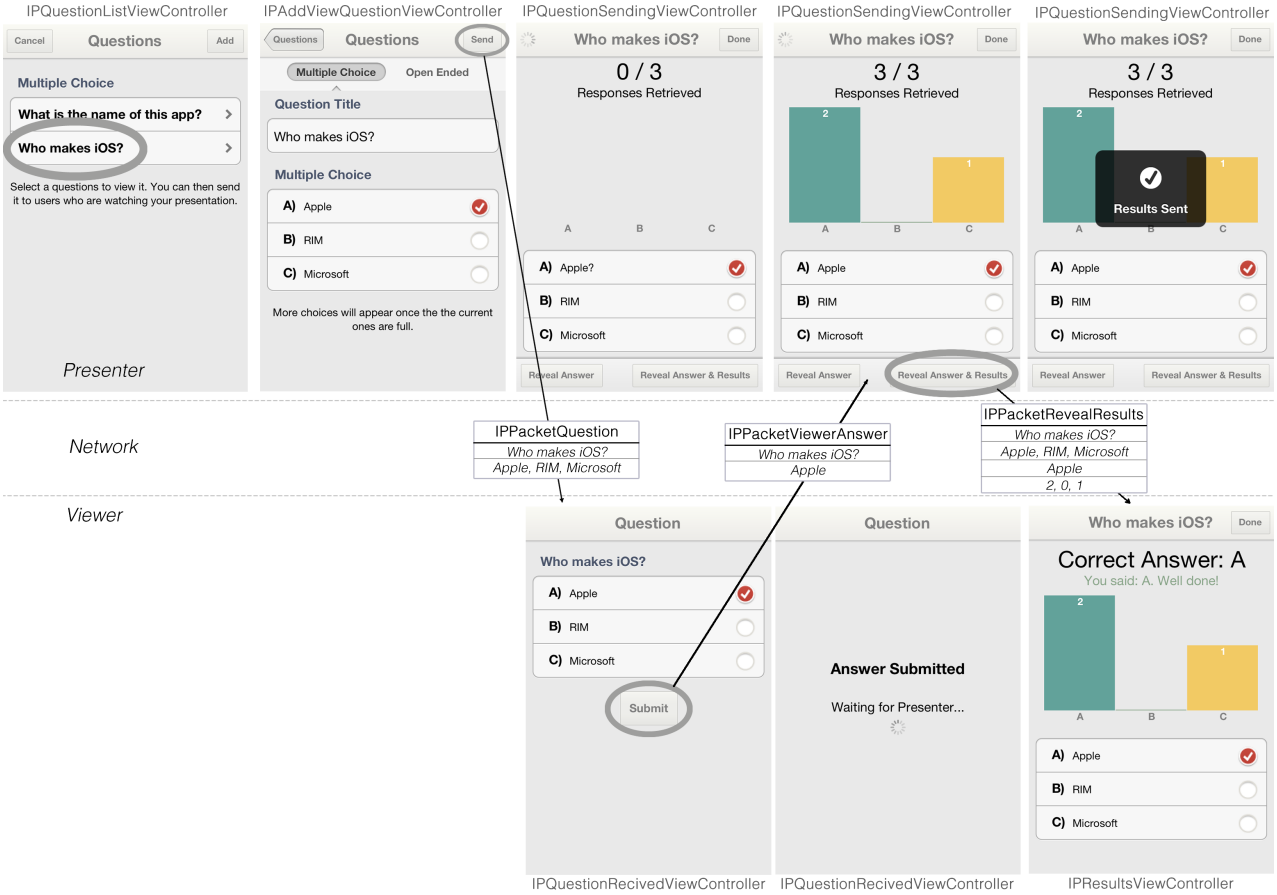


Figure 4.26: Questions and Answer device interaction

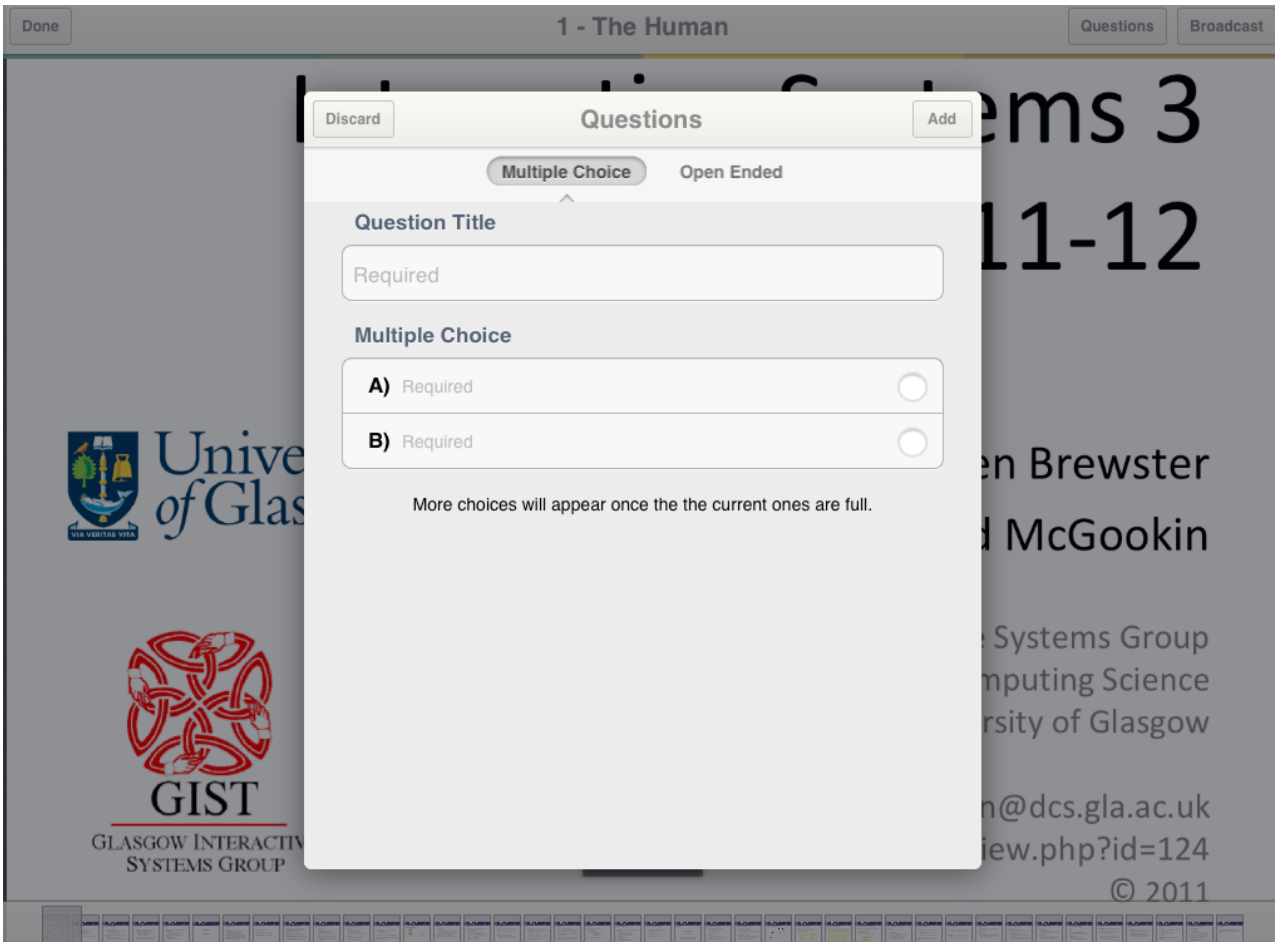


Figure B.2c: iOS 6

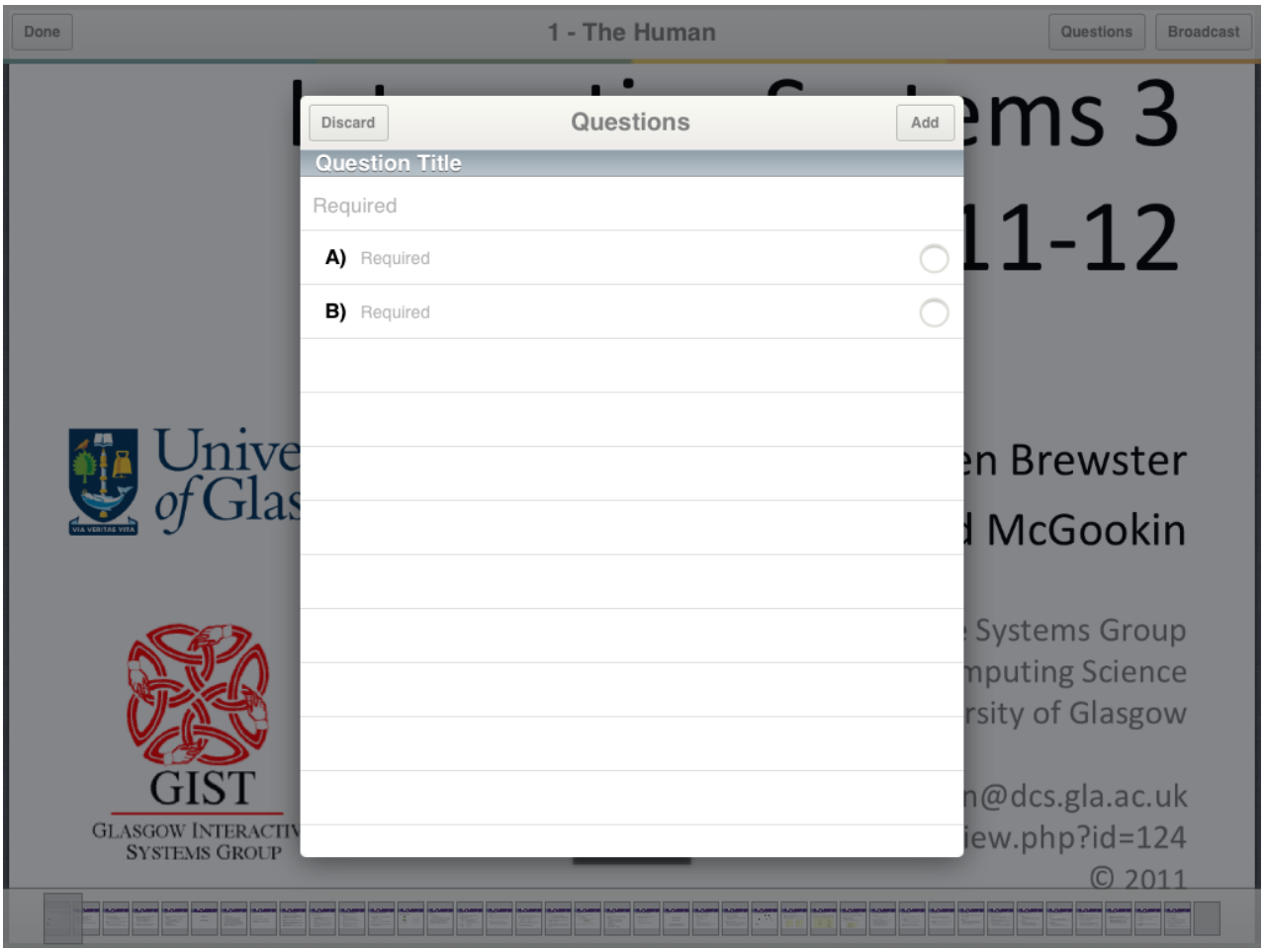


Figure B.2b: iOS 5

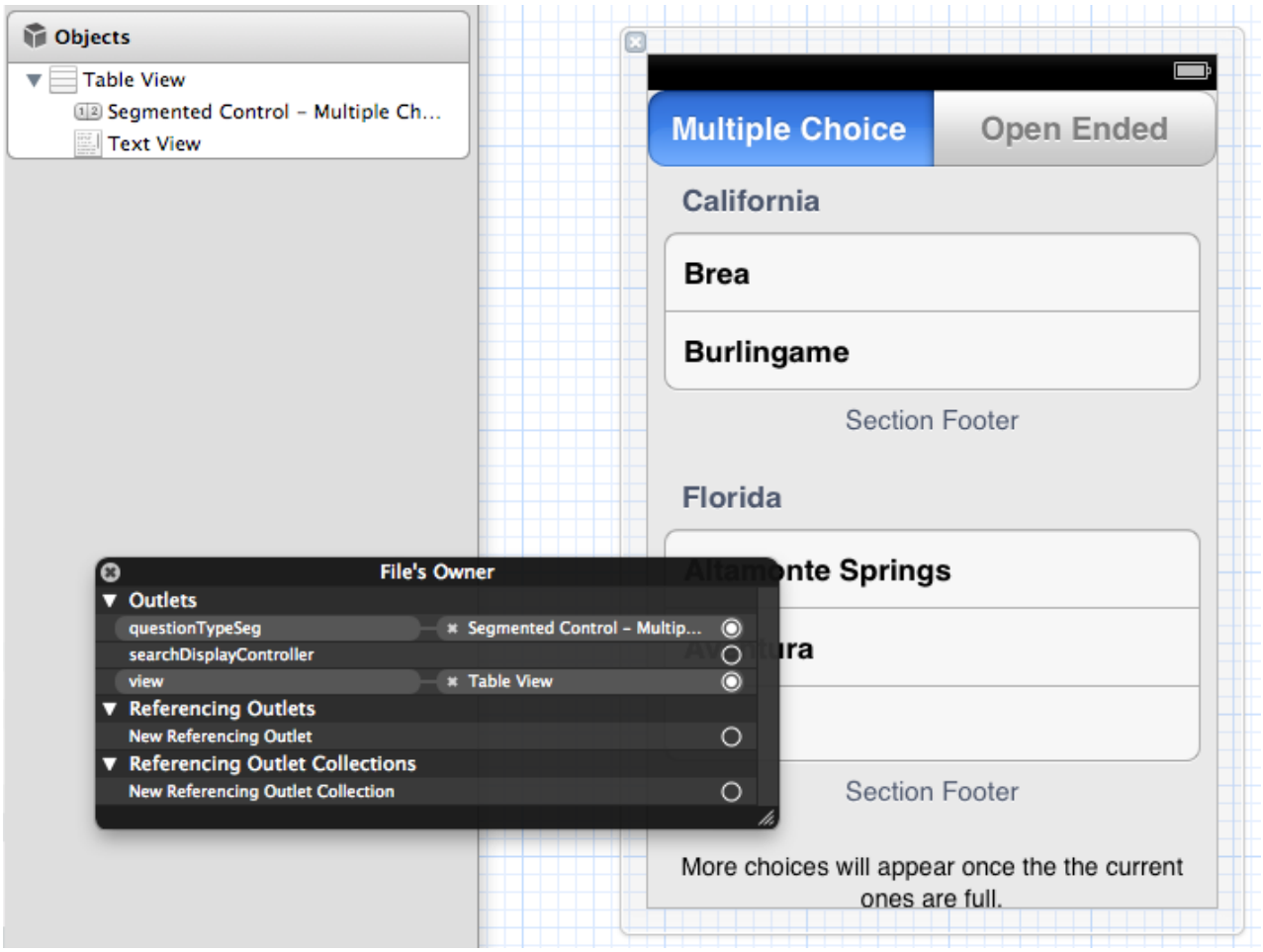


Figure B.2c: Interface file (XIB)

Answer Options	1 - High Priority	2	3	4	5	6	7	8	9	10 - Low Priority	Rating Average
Viewer raised questions	1	2	3	1	0	0	0	0	0	0	2.57
Save documents from a presentation	2	2	1	0	1	0	0	1	0	0	3.14
'Open In' support	2	1	0	0	2	0	1	1	0	0	4.14
Lock/Unlock presentation	0	1	2	1	0	0	2	0	0	1	5.14
Open ended questions	0	0	0	4	1	1	0	0	1	0	5.14
Passcode secured presentations	1	0	0	0	2	2	1	0	0	1	5.71
Annotation support	0	1	0	1	0	2	1	1	1	0	6.00
Document Syncing	1	0	0	0	1	0	1	2	1	1	6.86
Note taking via keyboard	0	0	0	0	0	1	1	2	2	0	8.00
Custom usernames	0	0	1	0	0	1	0	0	1	2	8.29

Figure 4.28: User additional feature rank

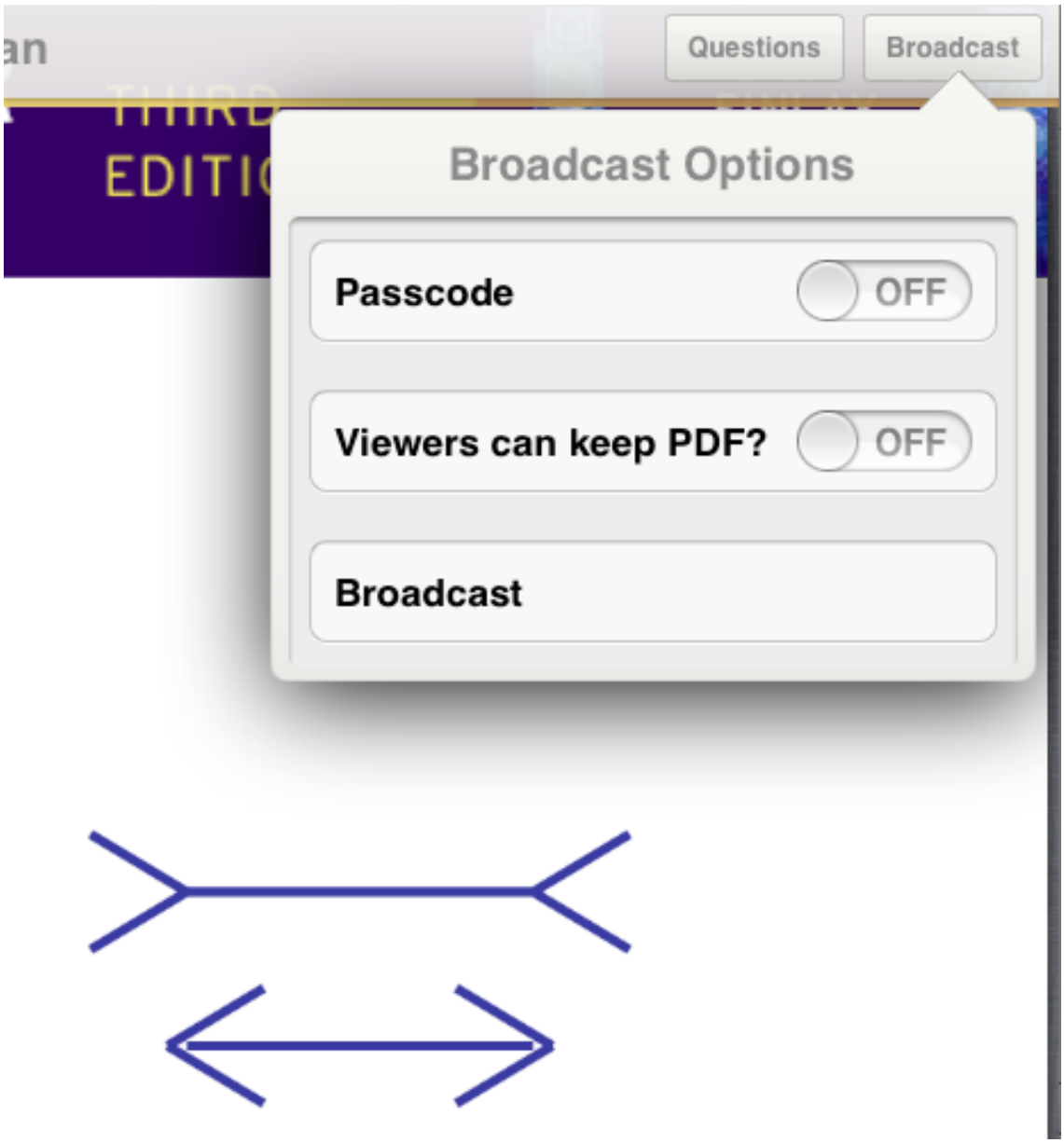


Figure 4.29a: Passcode disabled by default

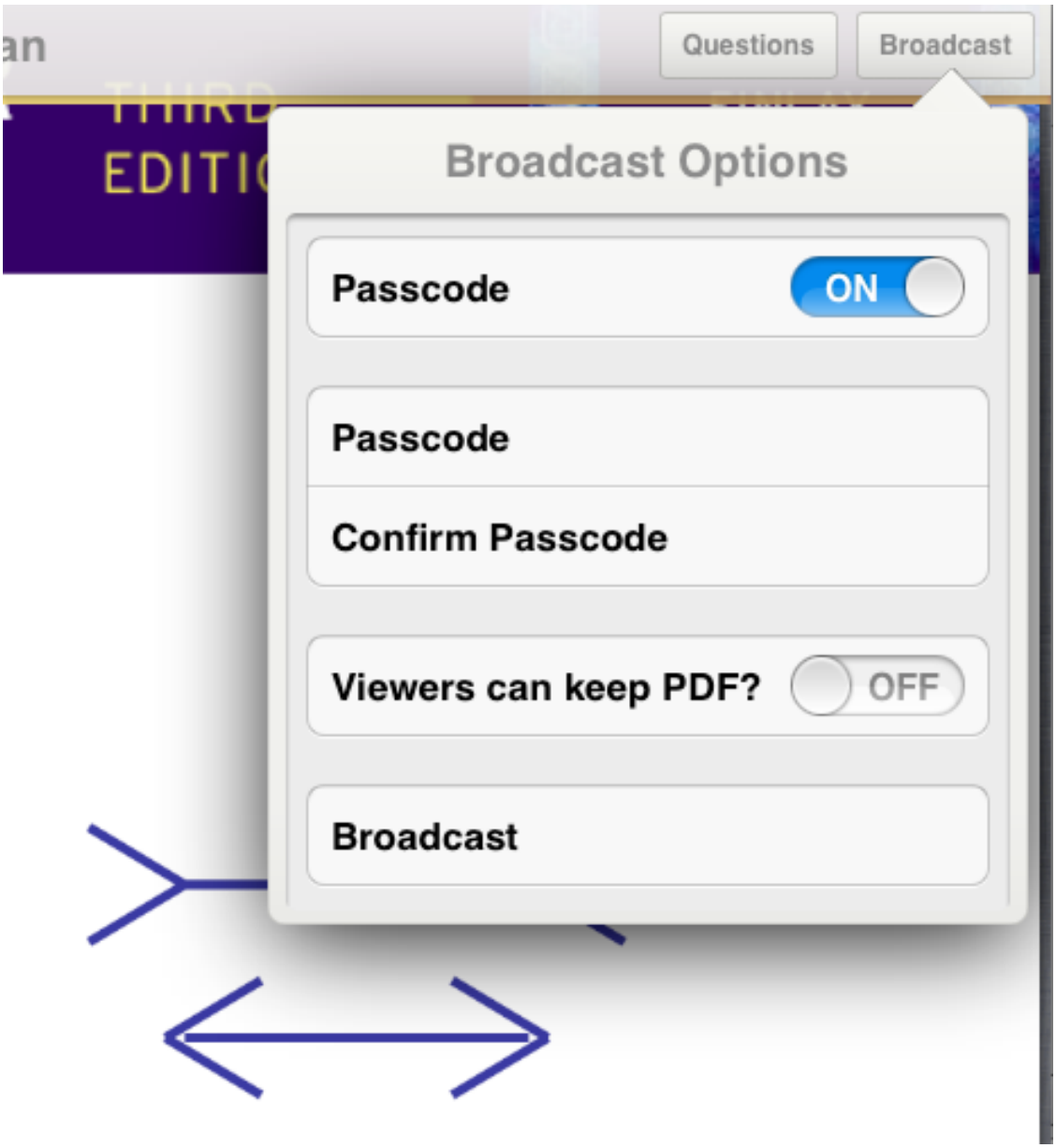


Figure 4.29b: Passcode enabled

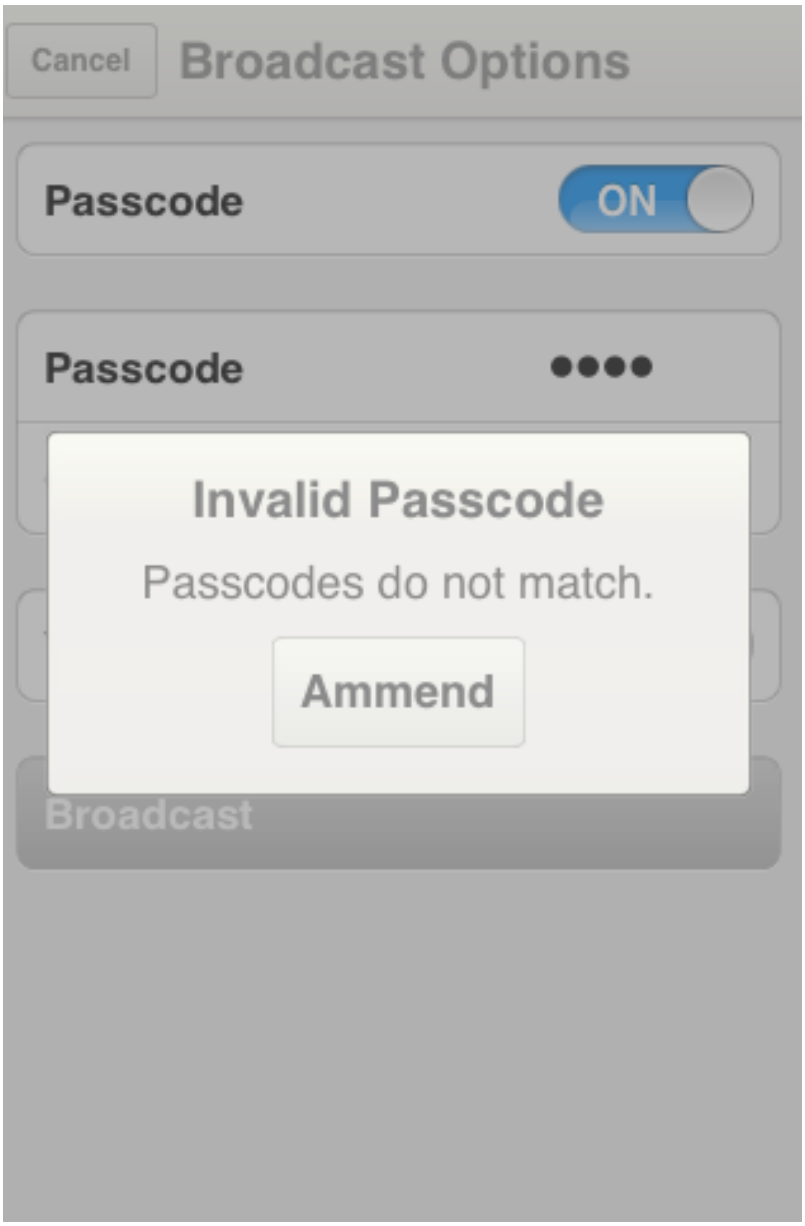


Figure 4.29c: Passcode validation

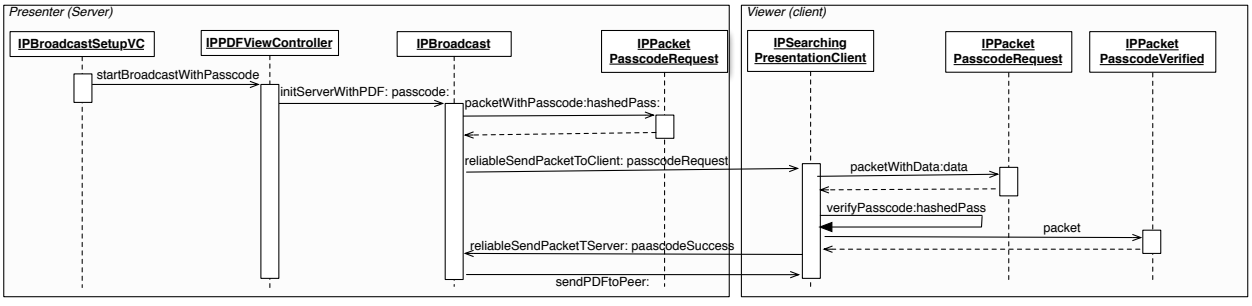


Figure 4.30: Passcode verification between devices

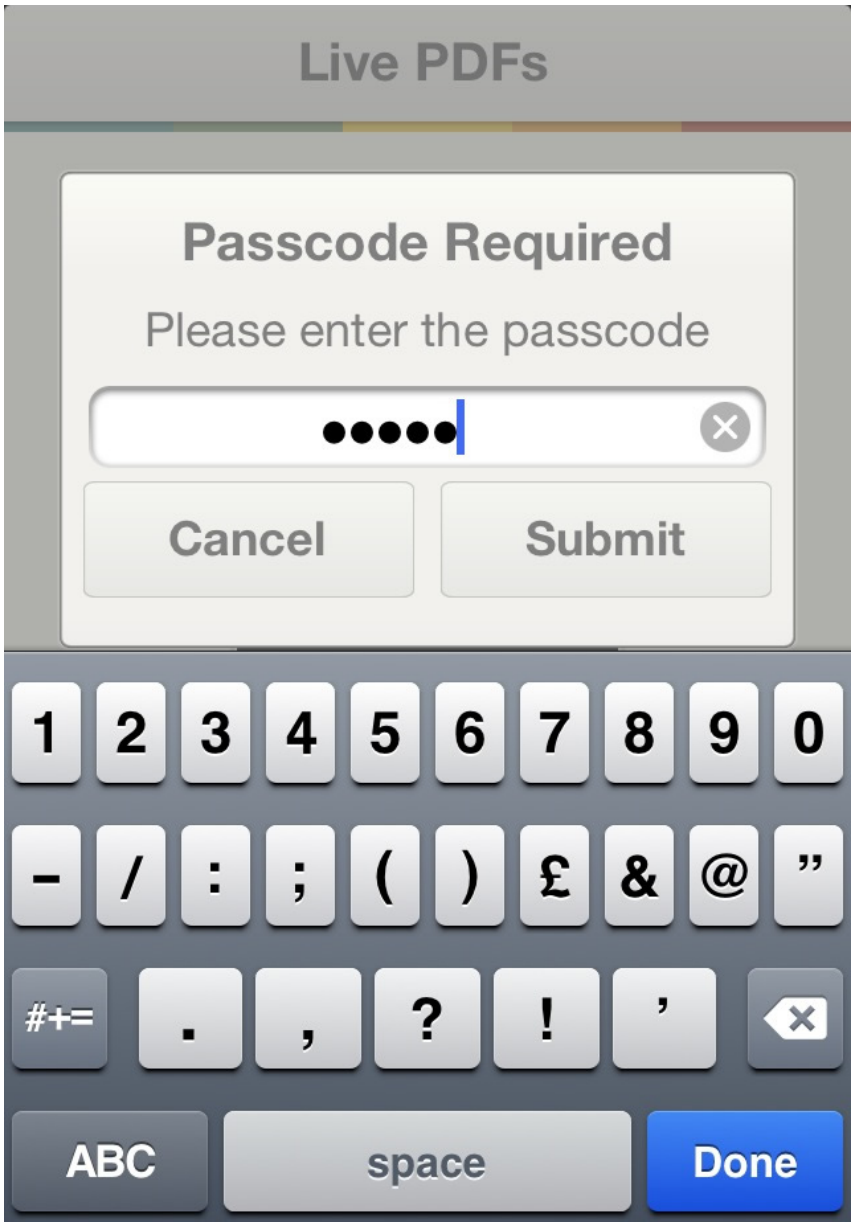


Figure 4.31a: Viewer passcode prompt

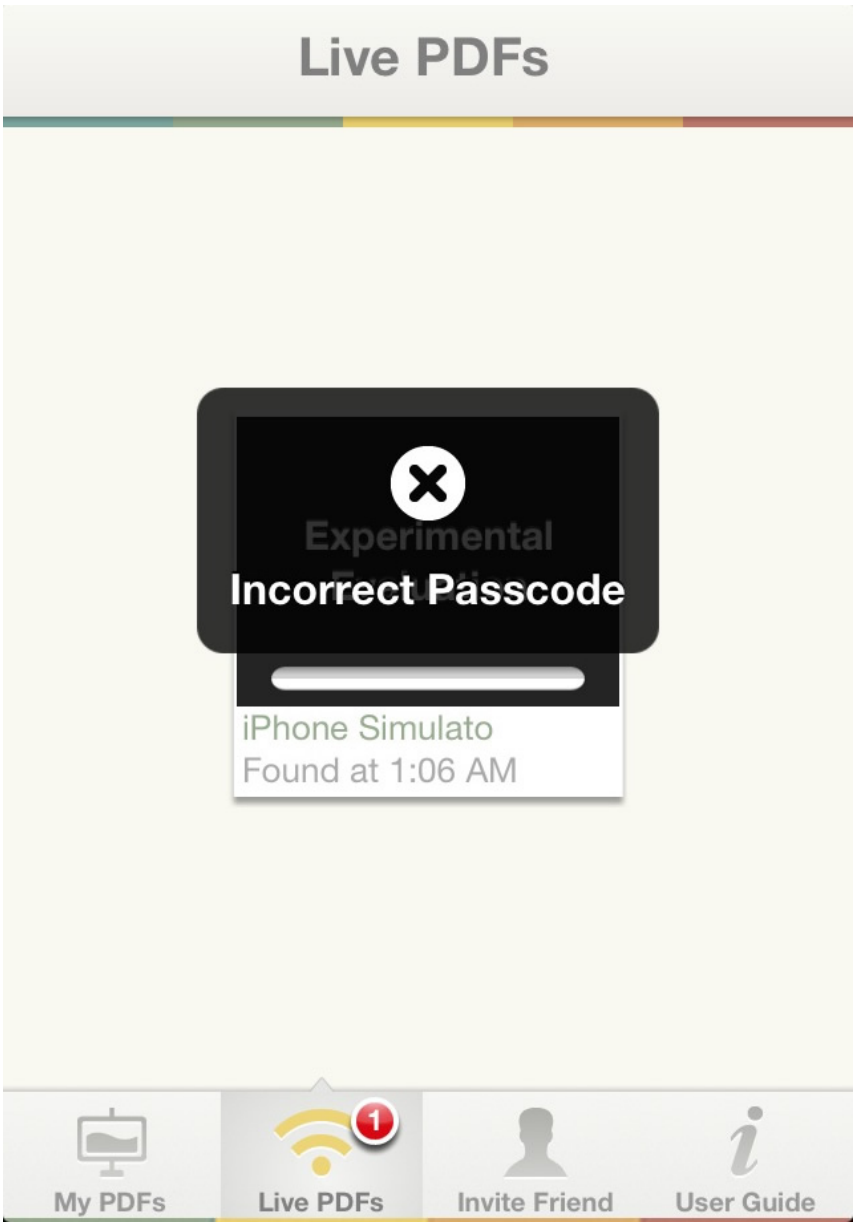


Figure 4.31b: Invalid passcode alert

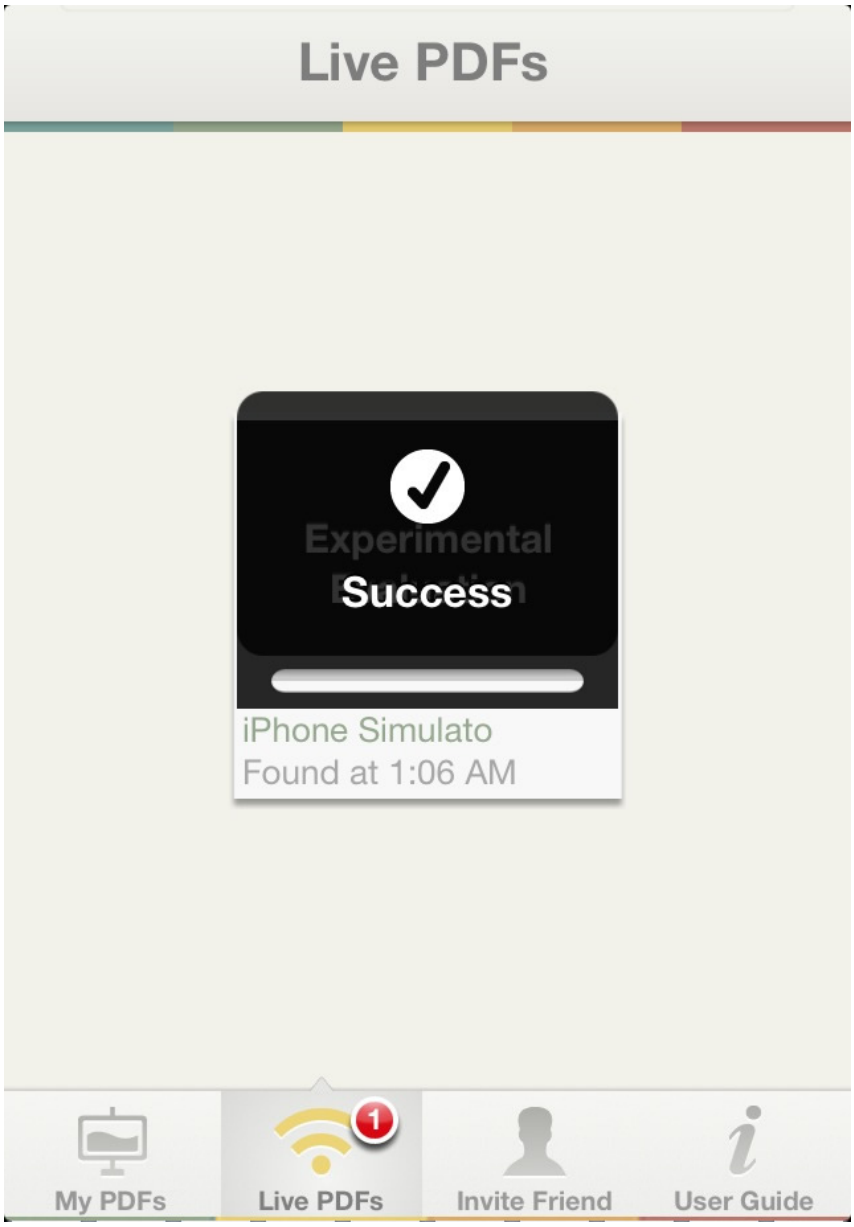


Figure 4.31c: Passcode verified successully

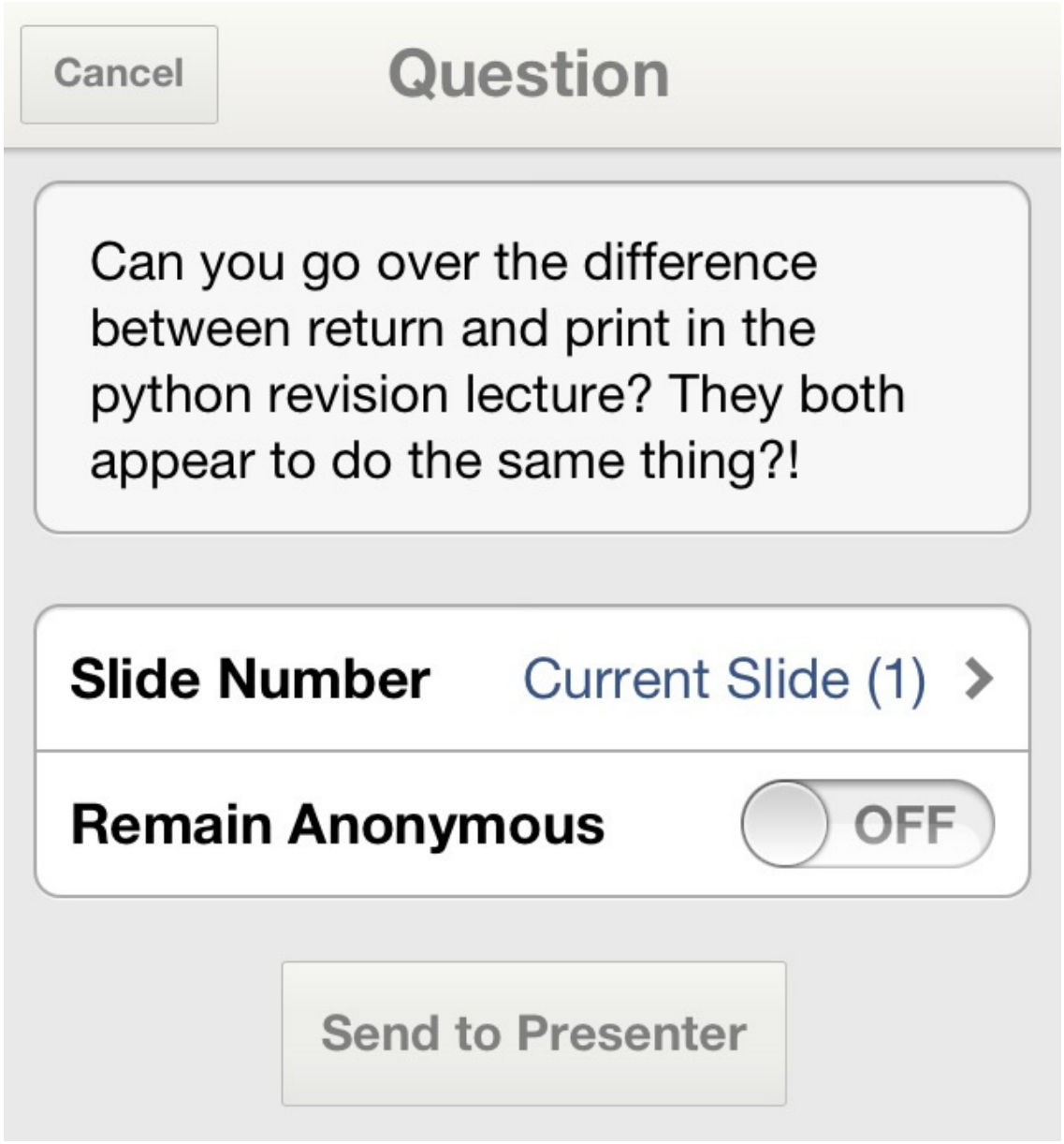


Figure 4.32a: iPhone

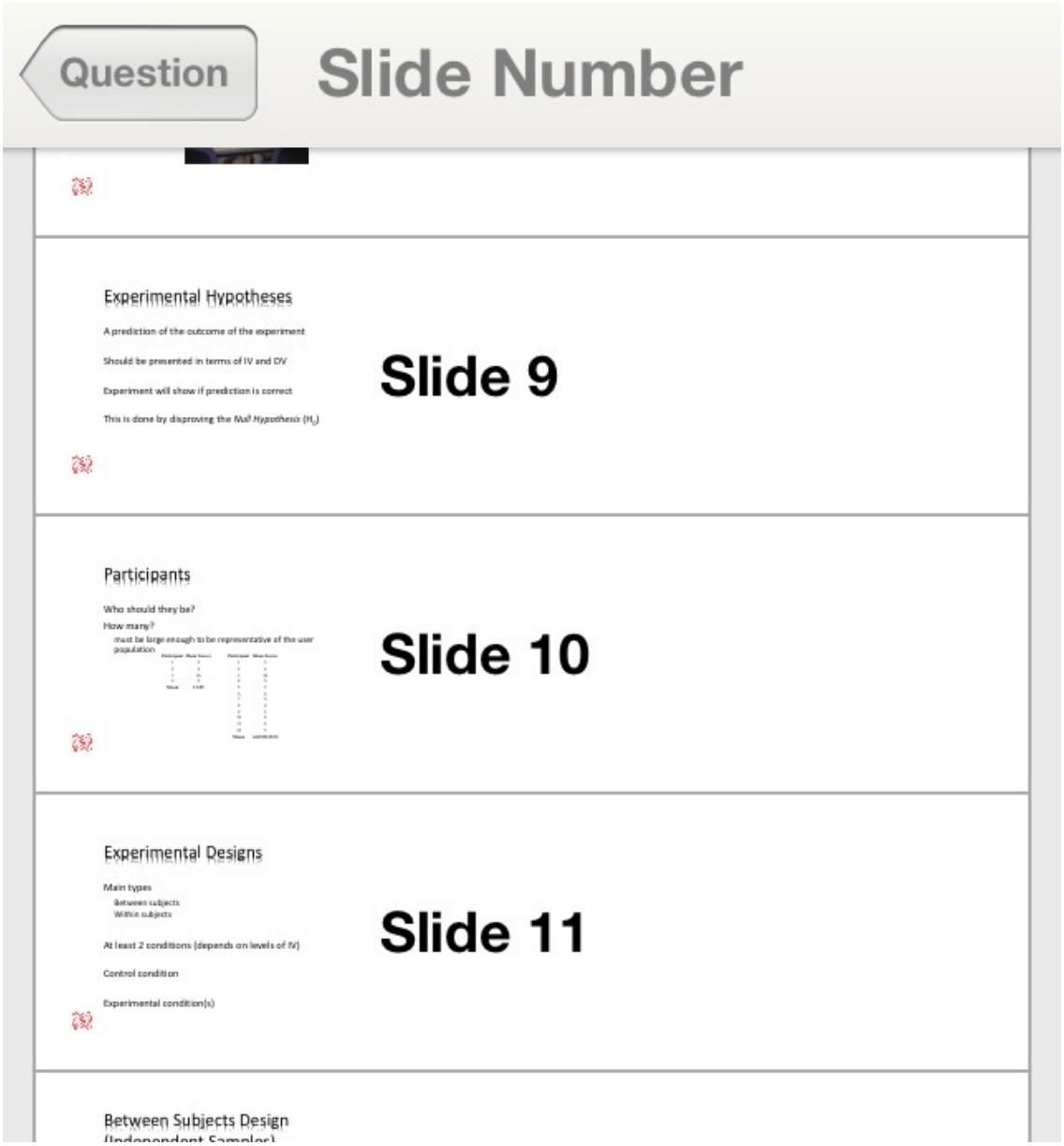


Figure 4.32b: Ability to include slide number

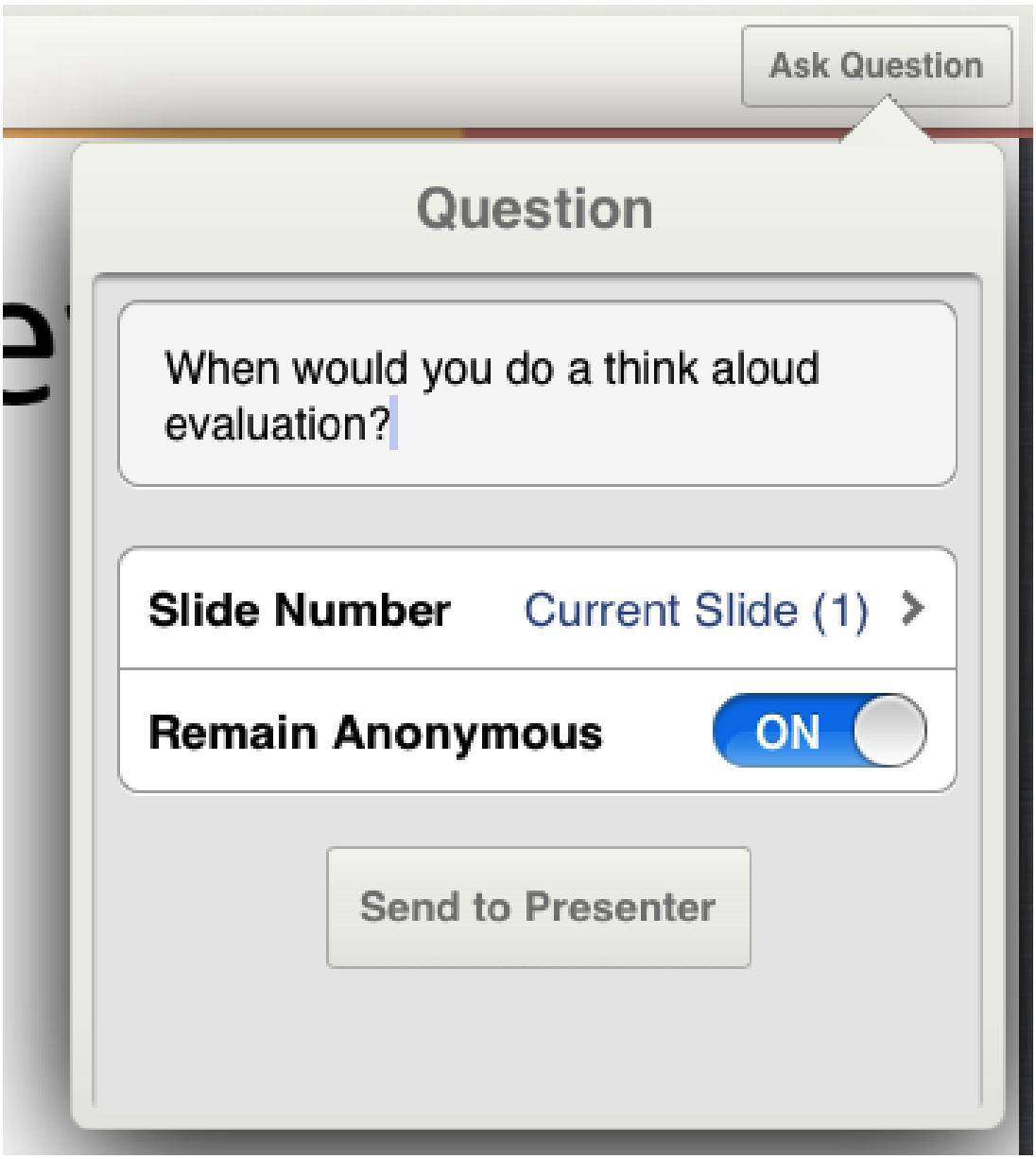


Figure 4.32c: iPad

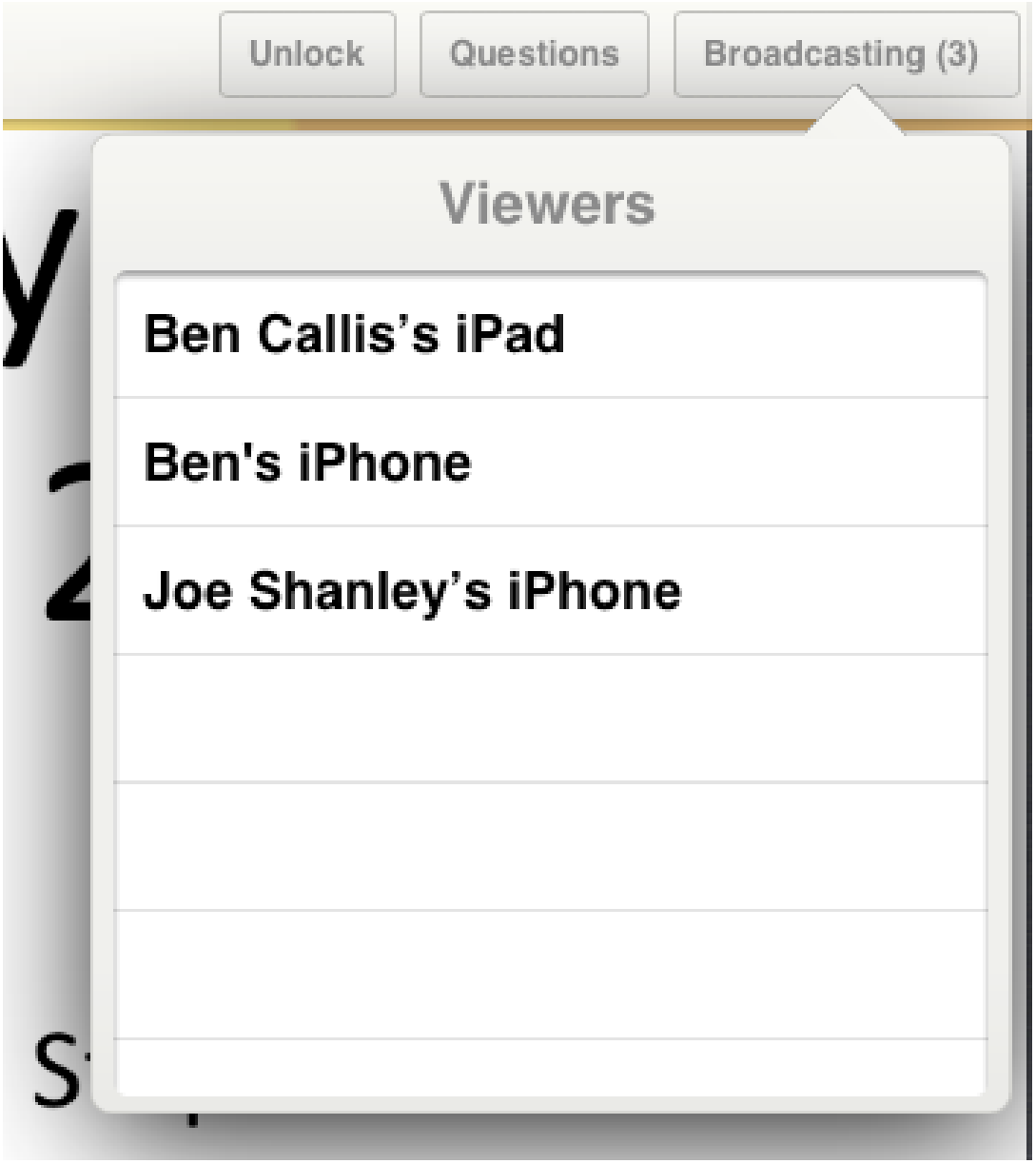
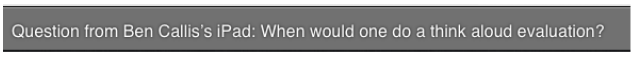
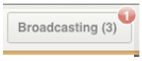


Figure 4.33: Viewer List

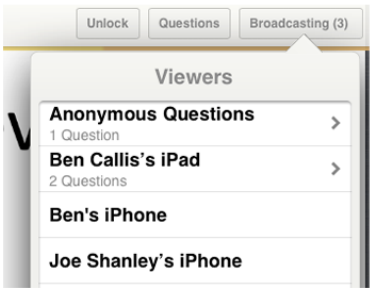


(a) Question notification

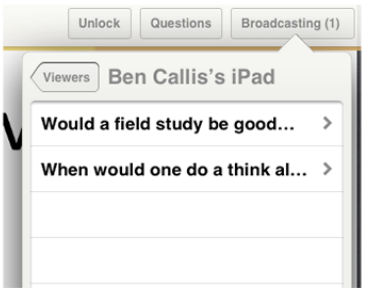
(b) Annonymous notification



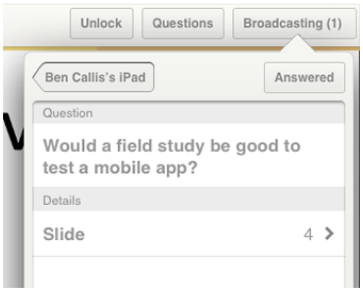
(c) badge



(d) Viewer list with questions



(e) List of user questions



(f) Question details

Figure 4.34: Views to allow presenter to access viewer's questions

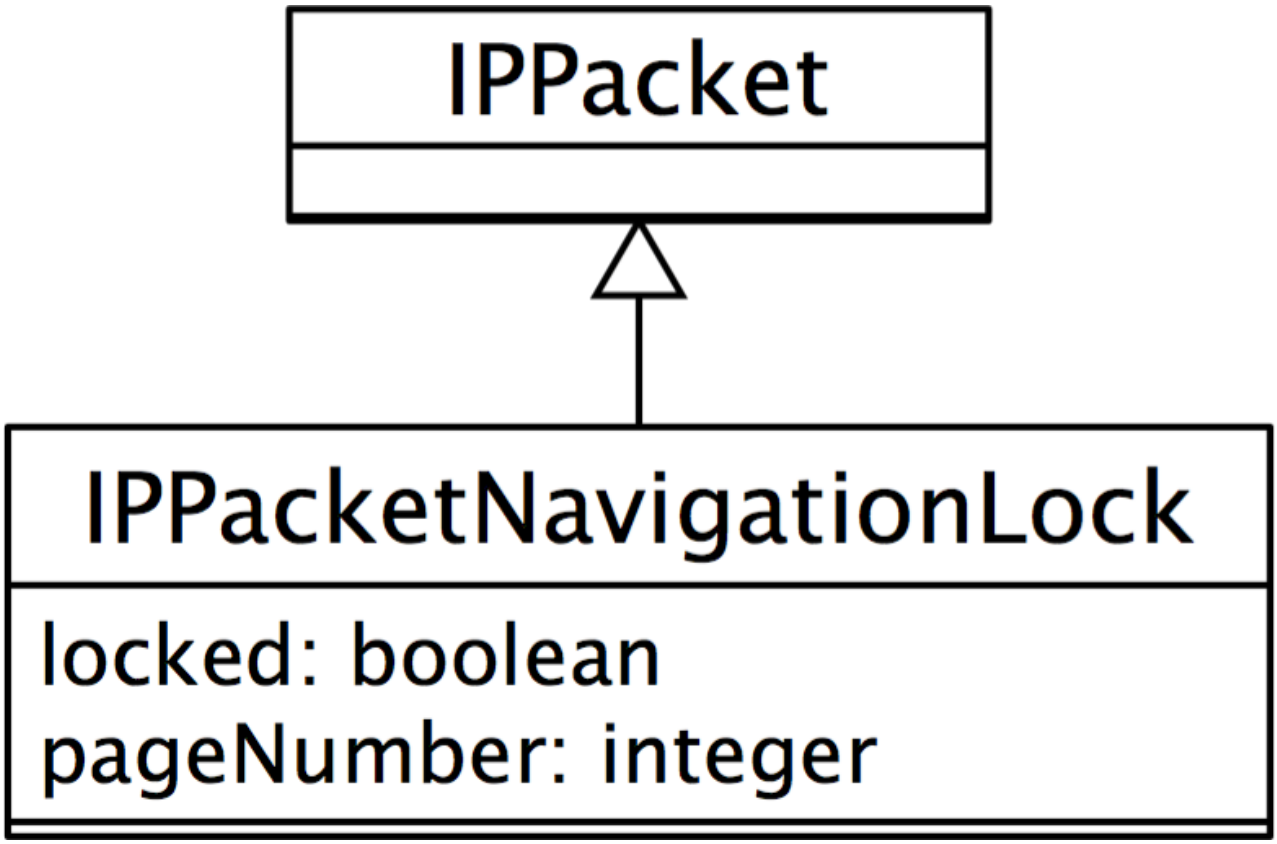


Figure 4.35: Lock packet class

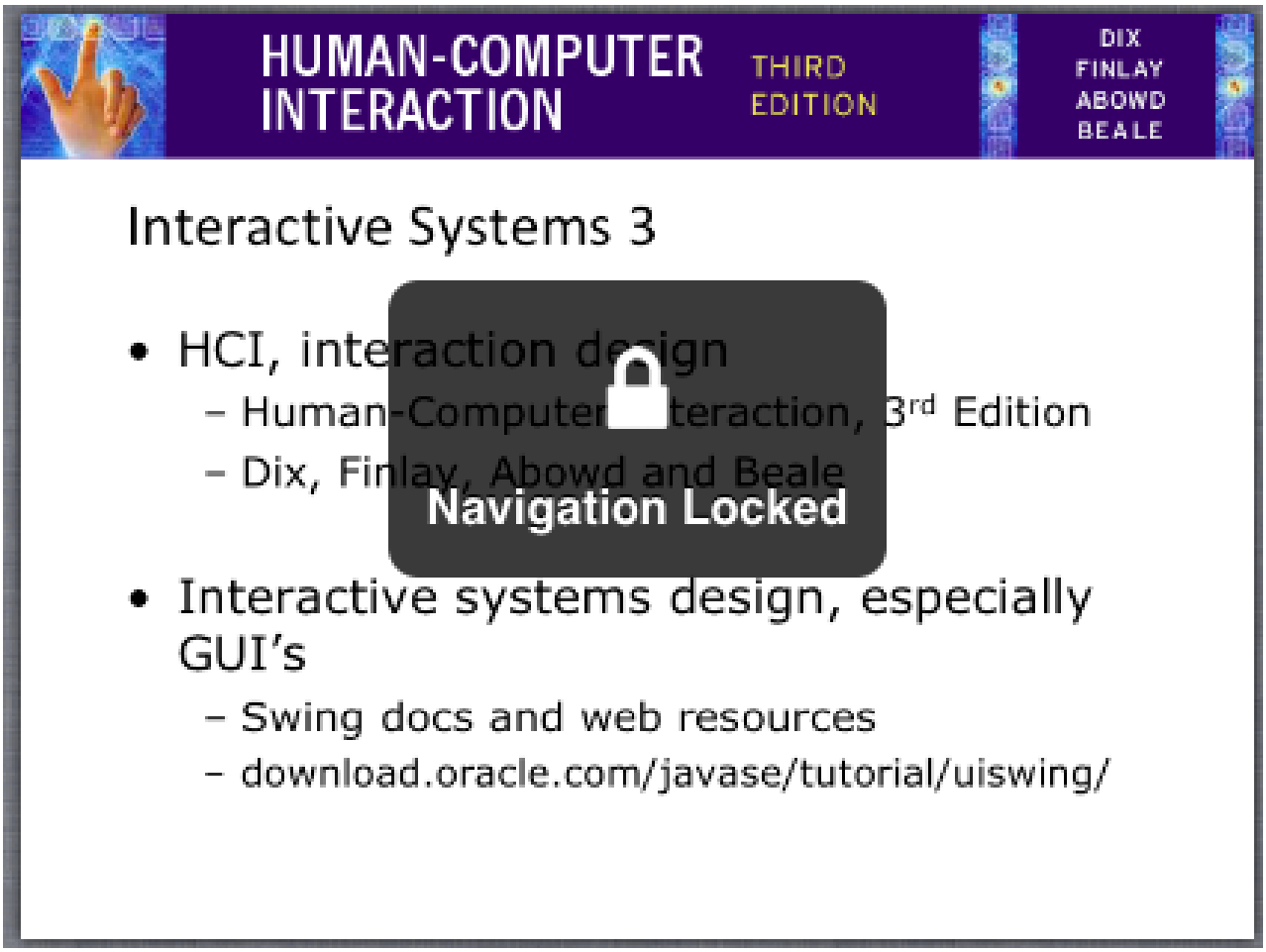


Figure 4.36: Locked alert

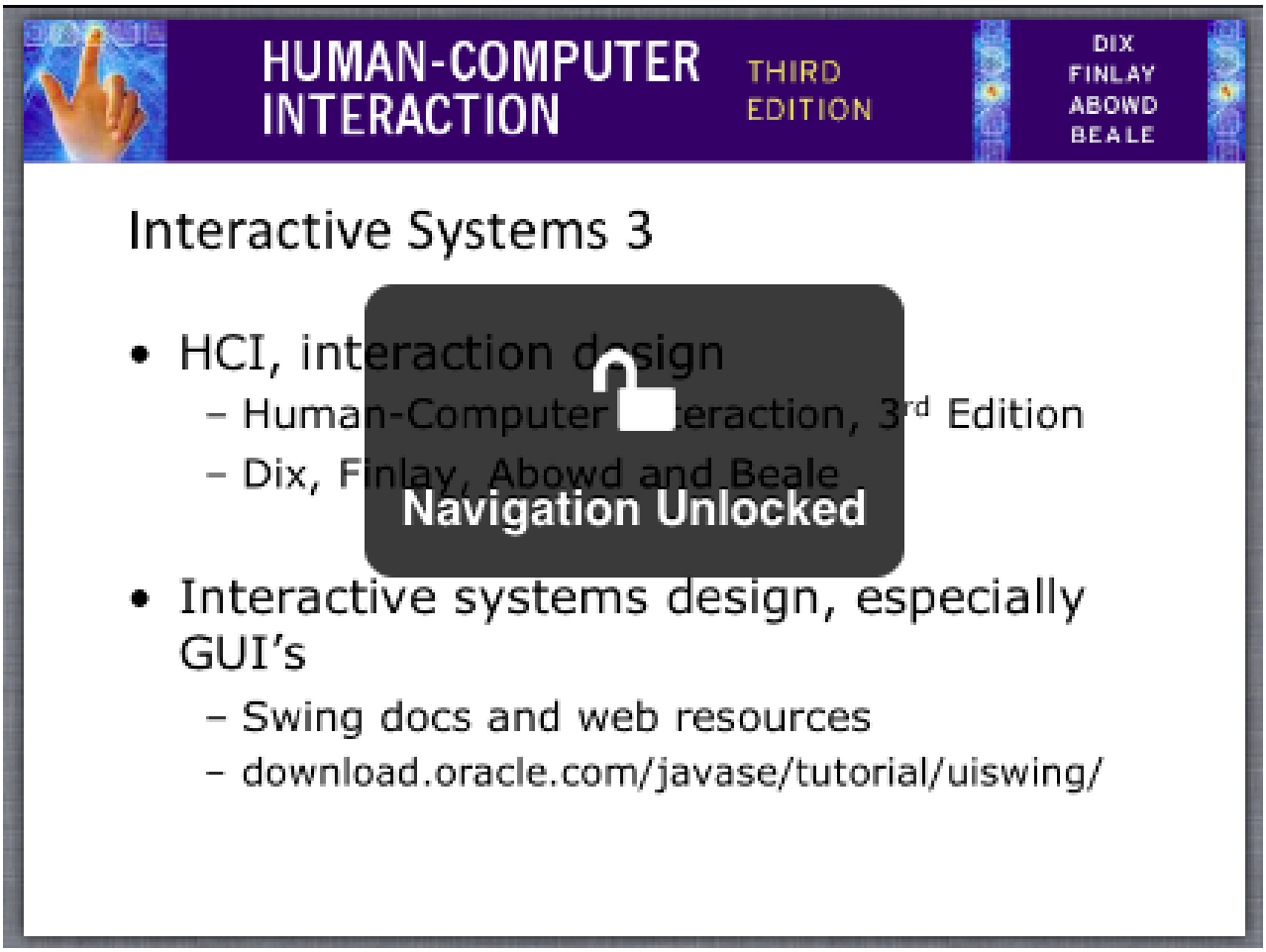


Figure 4.37: Unlocked alert

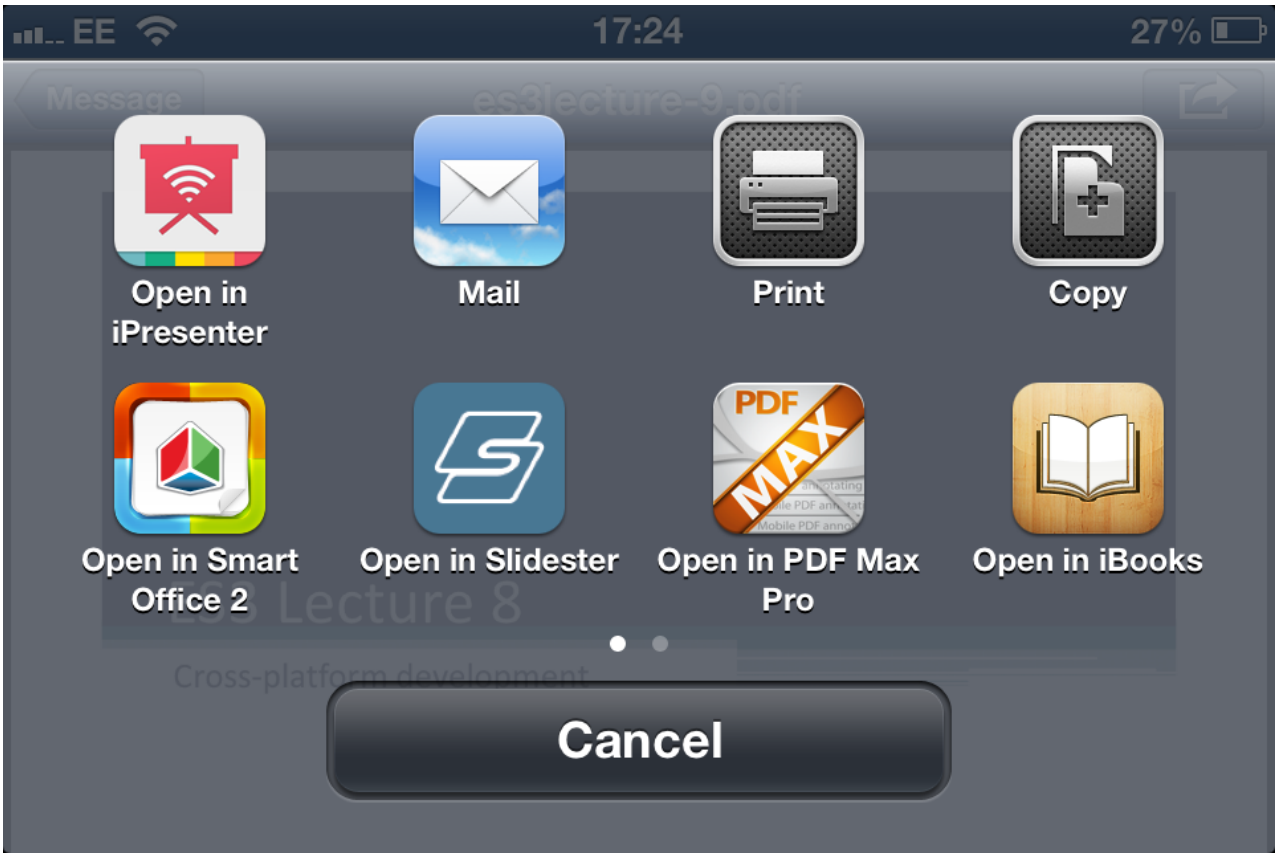


Figure 4.38: 'Open in' support

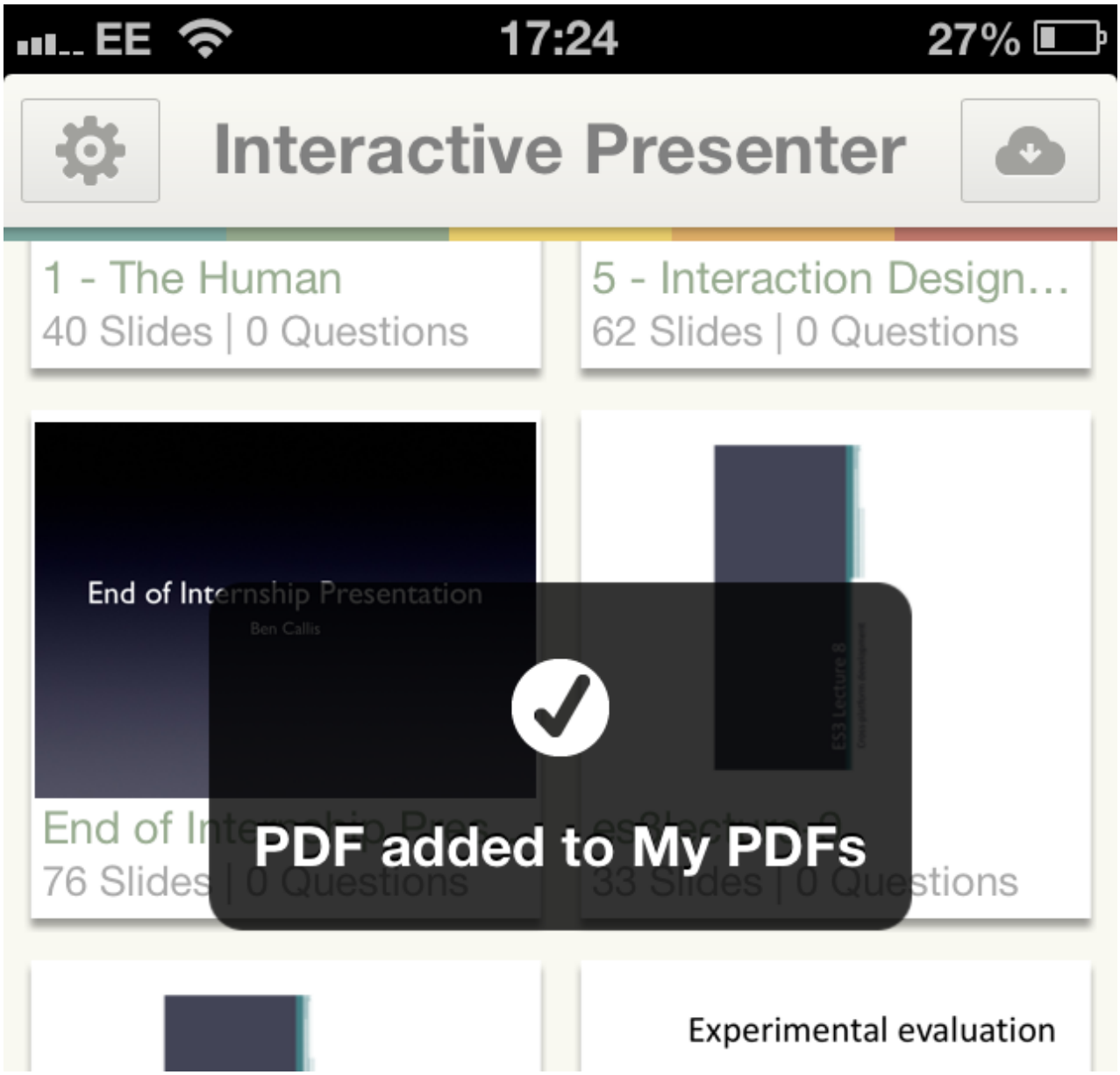


Figure 4.39: PDF successfully added

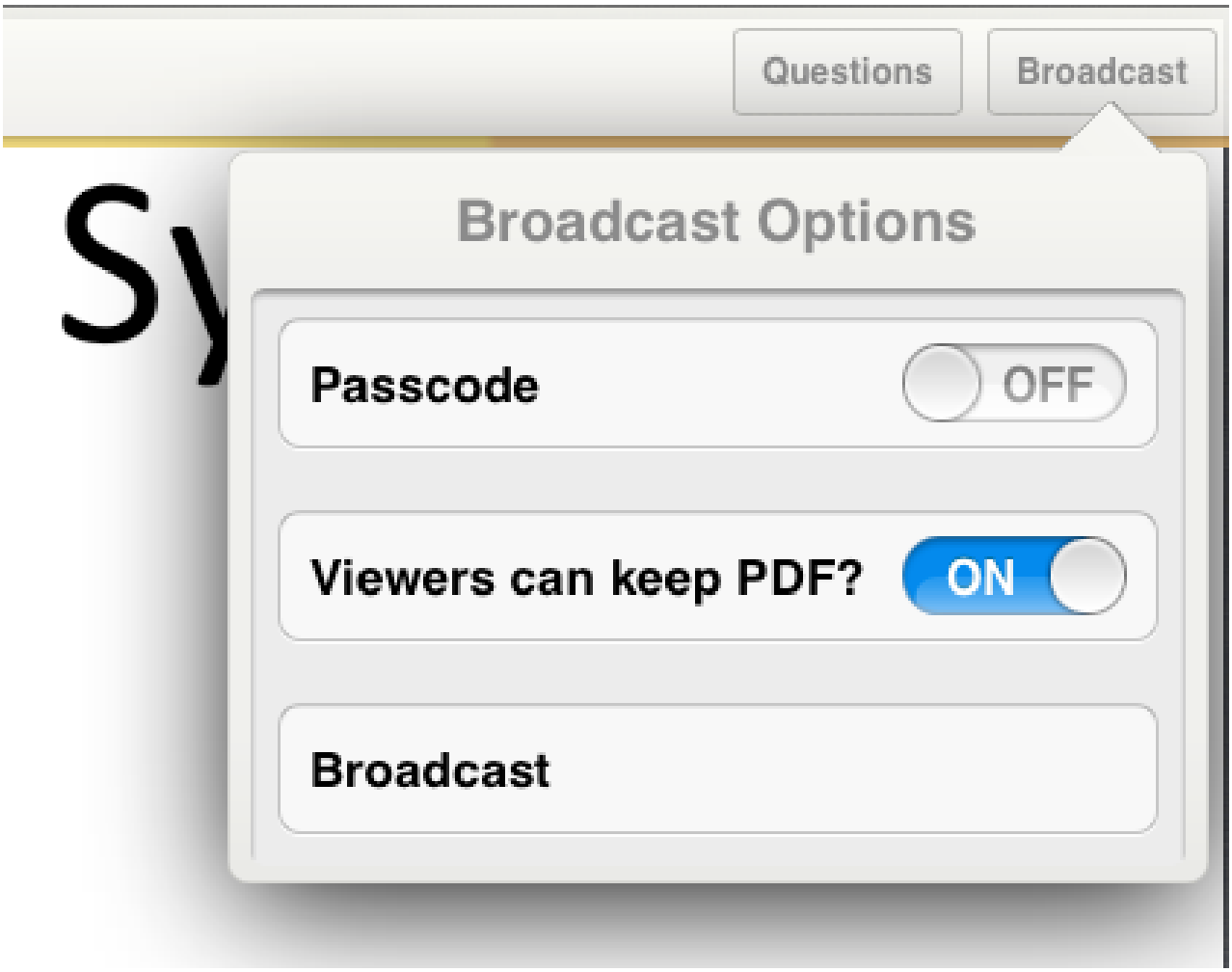


Figure 4.40a: Permission set in broadcast options

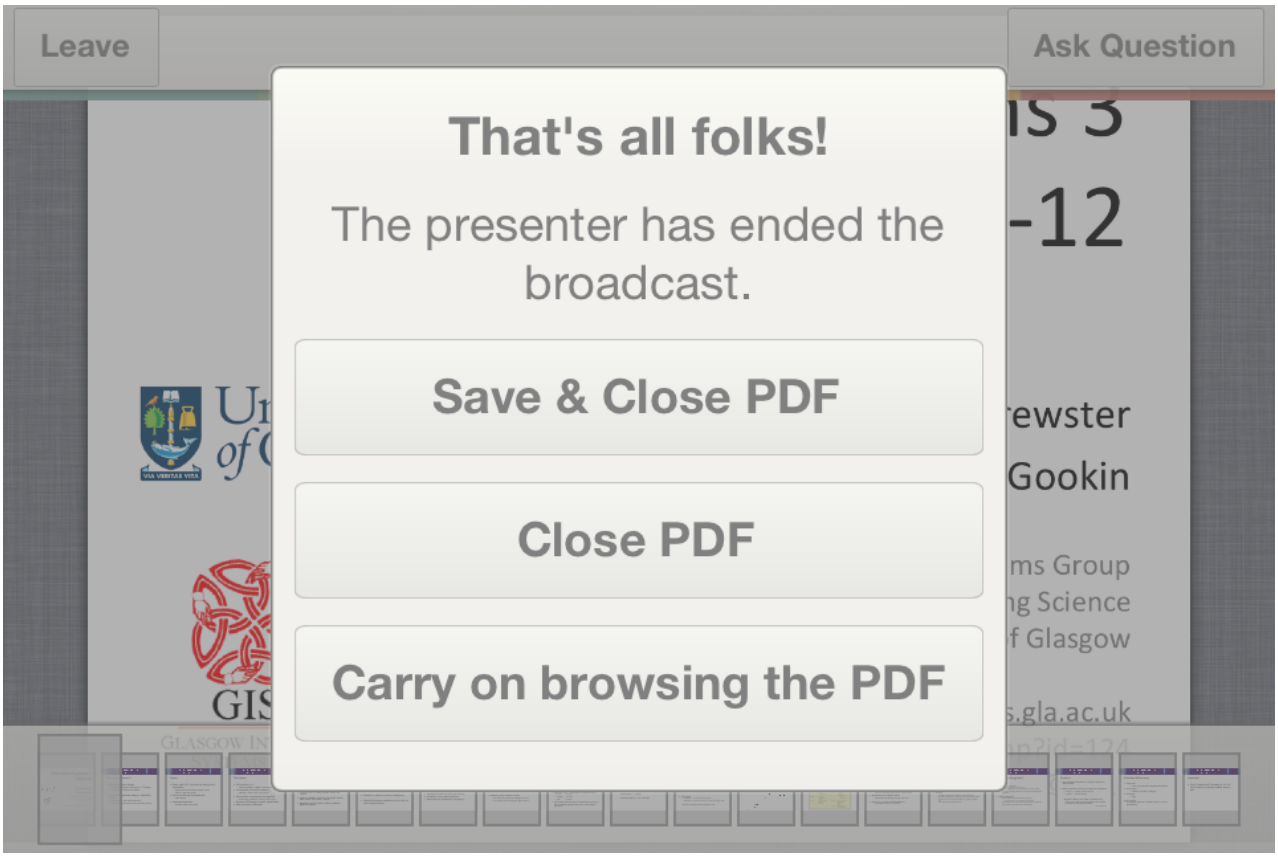


Figure 4.40b: Save PDF when presenter leaves

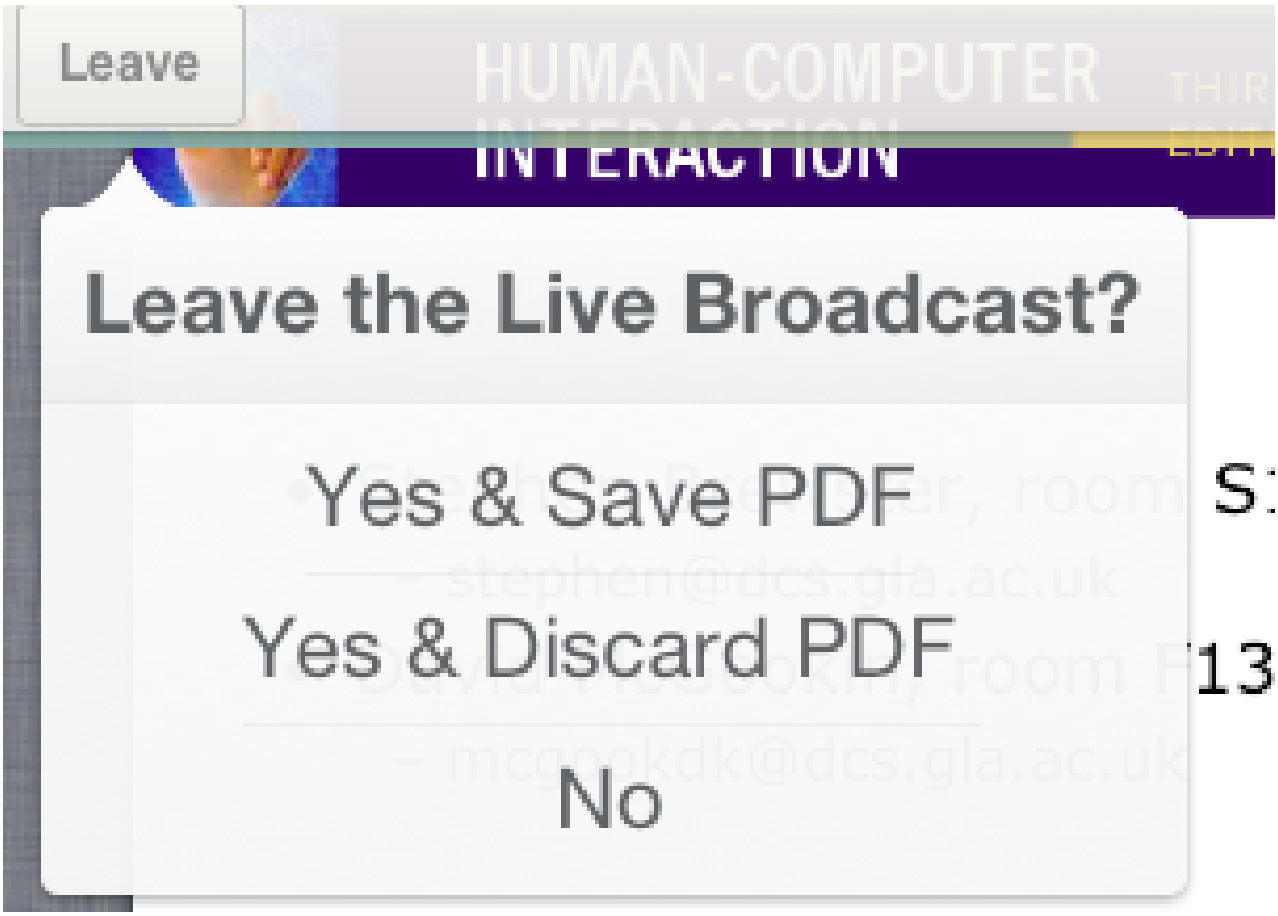


Figure 4.40c: Save PDF when viewer leaves

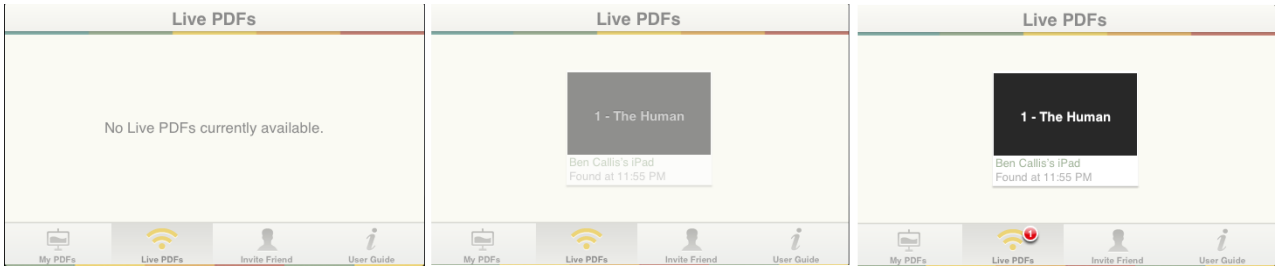


Figure 4.41: Live PDF animation

Figure 4.42: Part of the user guide included within the application

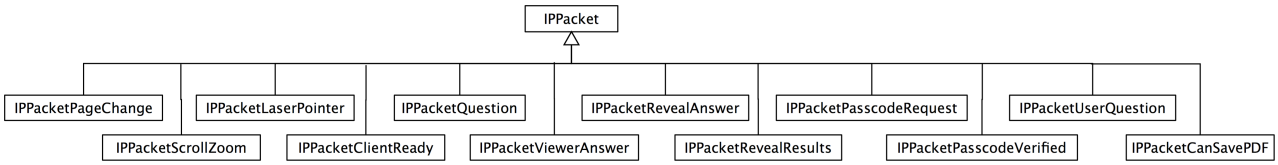


Figure 4.43: 12 packets used in the application

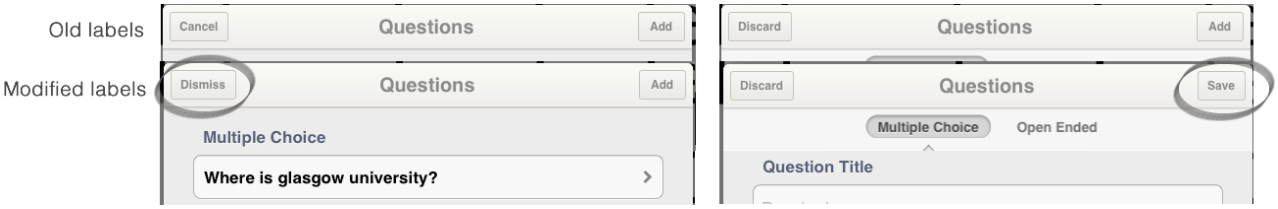


Figure 5.1: Alterations to the question button labels based on feedback.

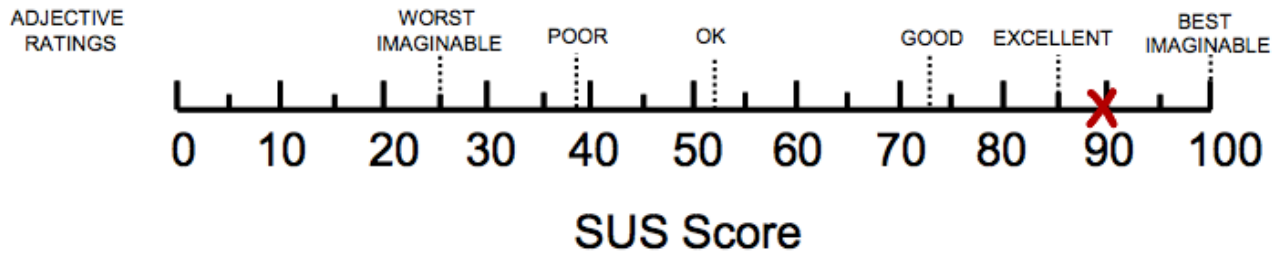


Figure 5.2: The application received an SUS score of 89.78 denoting 'Excellent usability'.

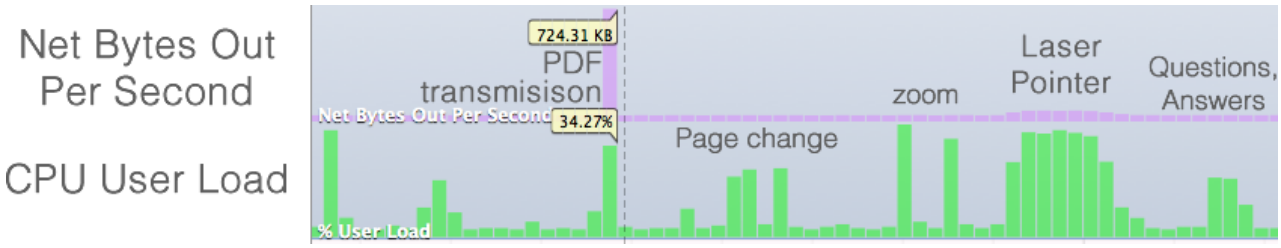


Figure 5.3a: Outgoing network and CPU usage

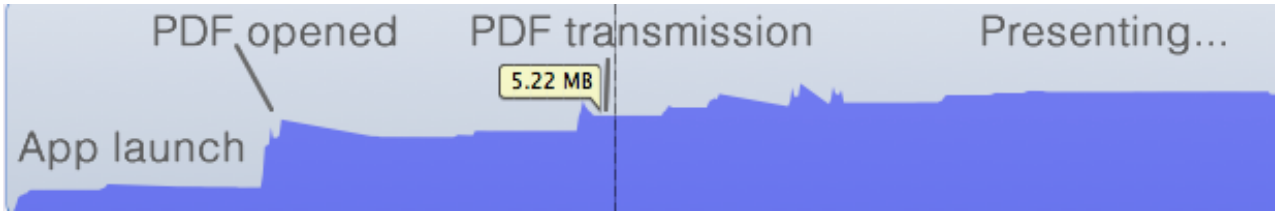


Figure 5.3b: Memory usage

Figure B.1: Devices capable of running iOS 5

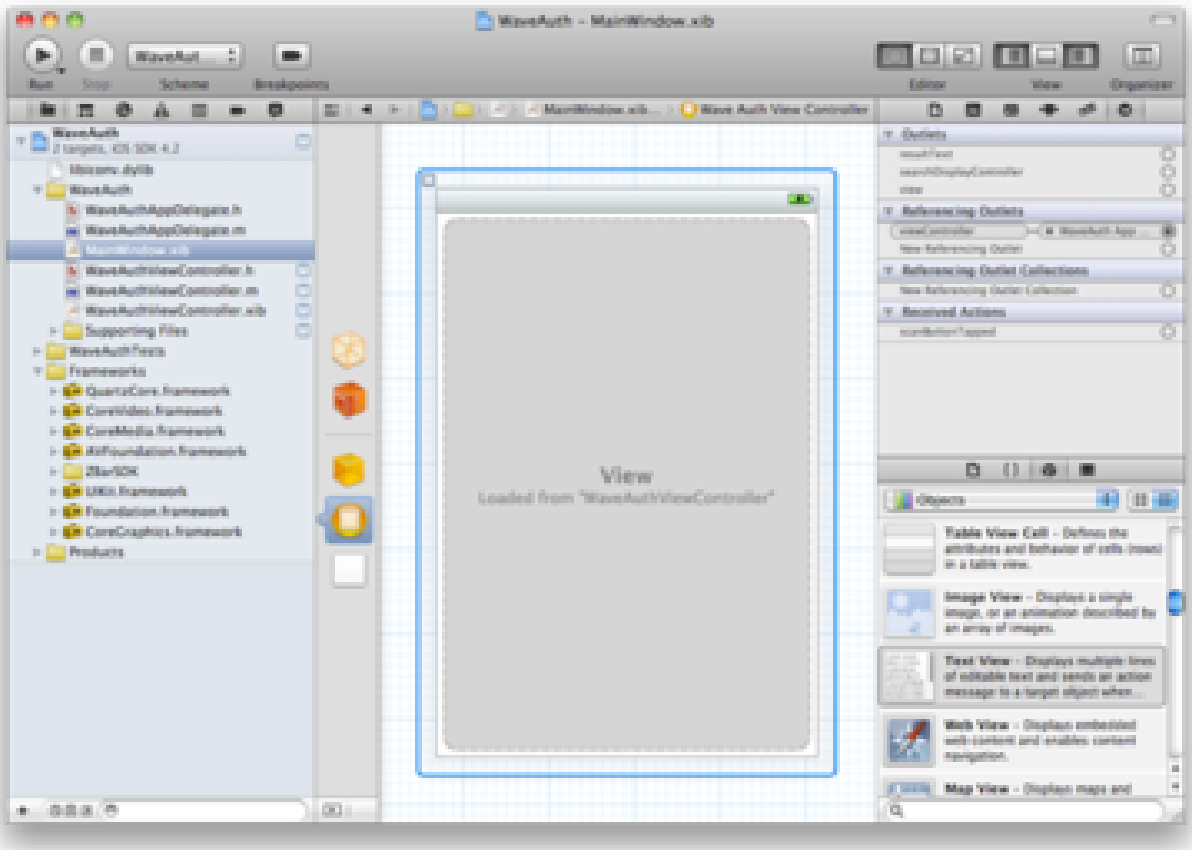


Figure B.2c: Xcode IDE

Figure B.2b: iOS Simulator

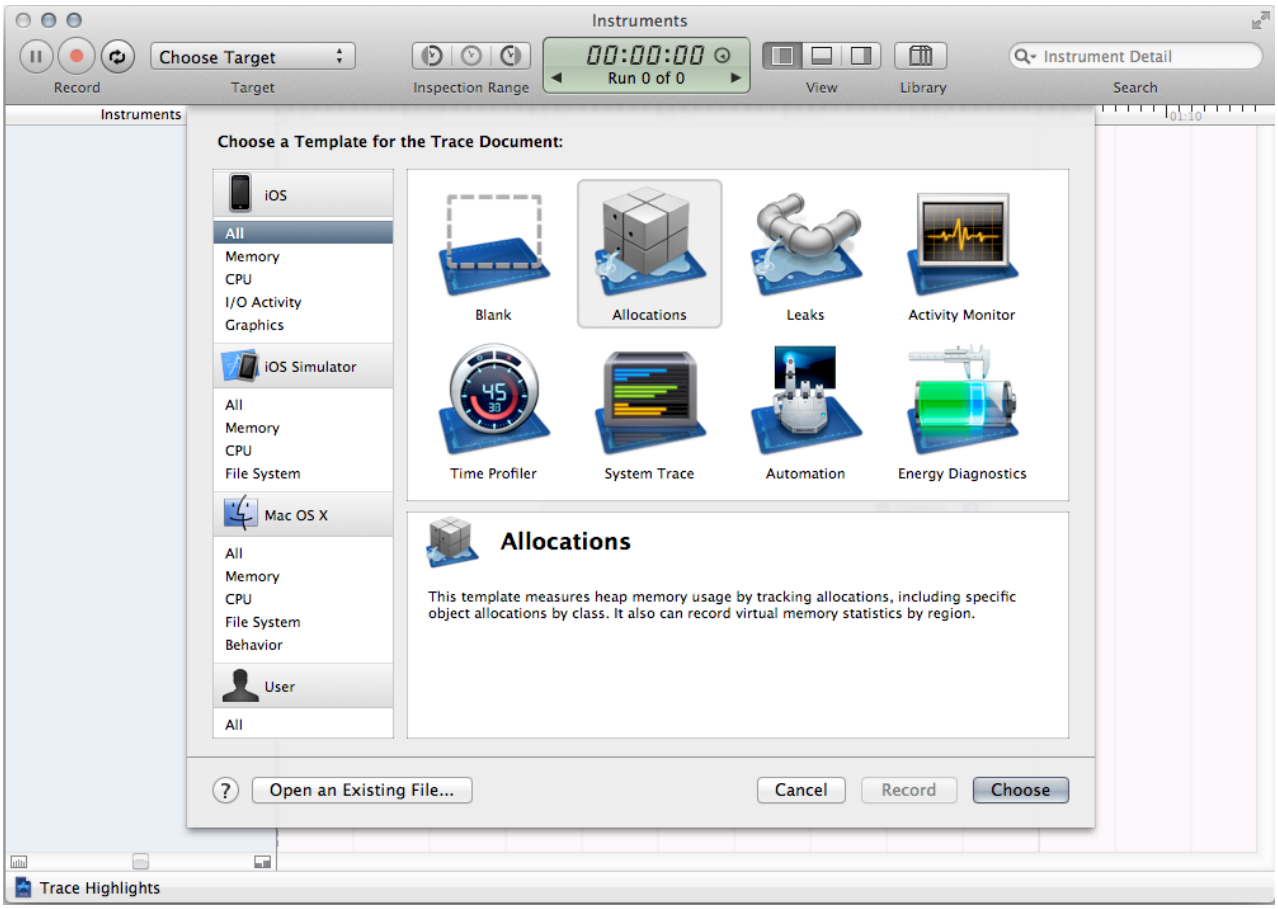


Figure B.2c: Instruments analysing tool

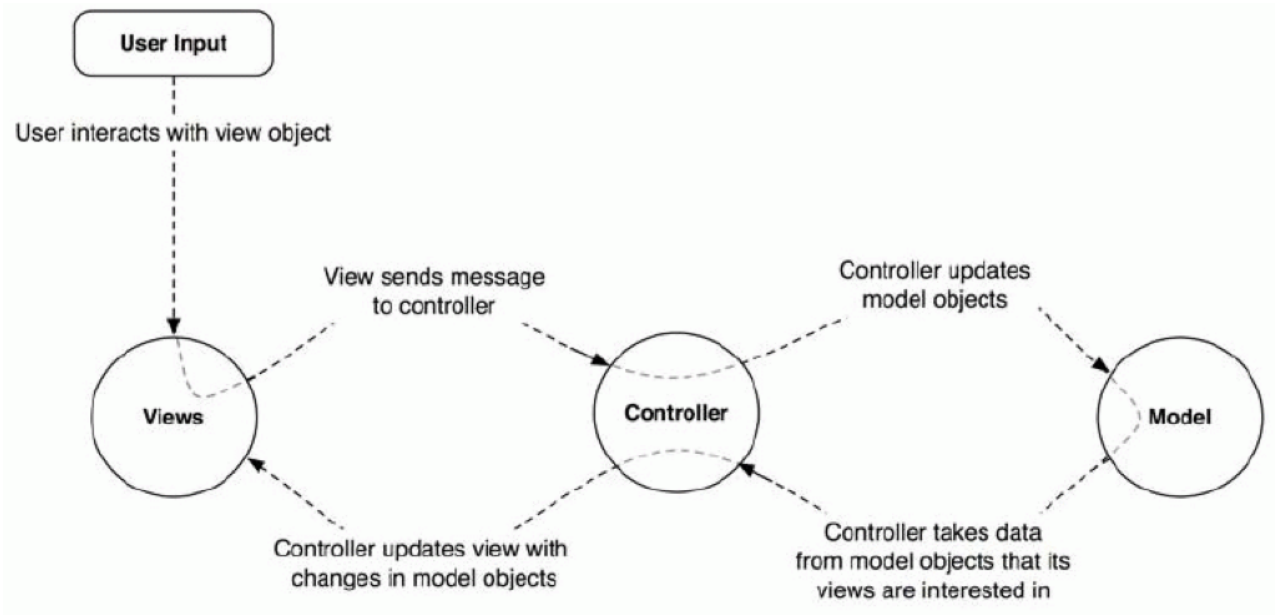


Figure B.3a: Modal-View-Controller design pattern [8]

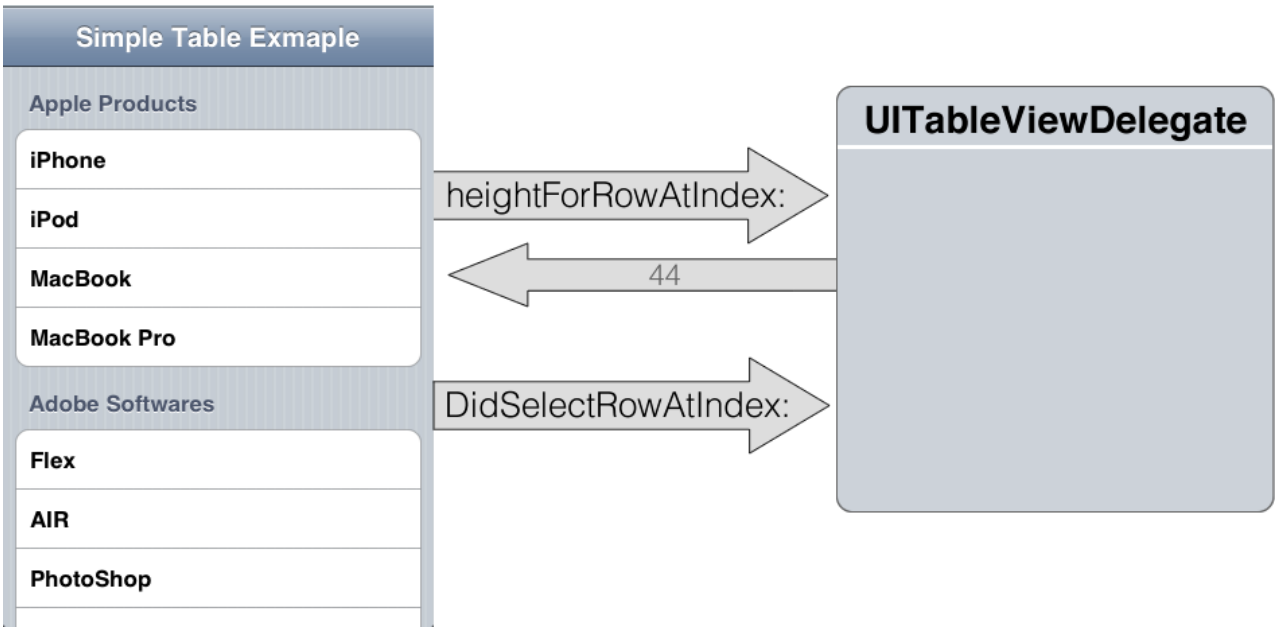


Figure B.3b: TableView delegation example