A group design project report submitted for the award of
MEng Computer Science

Supervisor: Dr Mike Wald
Examiner: Dr Tim Chown

noteEd - A web-based lecture capture system

by  Liam Fernando Alexander Horn Oliver Parson
    Matthew Porter Oliver Wells

    December 9, 2009
Electronic capture and playback of lectures has long been the aim of many academic projects. Synote is an application developed under MACFoB (Multimedia Annotation and Community Folksonomy Building) project to synchronise the playback of lecture materials. However, Synote provides no functionality to capture such multimedia. This project involves the creation of a system called noteEd, which will capture a range of multimedia from lectures and make them available to Synote.

This report describes the evolution of the noteEd project throughout the design and implementation of the proposed system. The performance of the system was checked in a user acceptance test with the customer, which is discussed after screenshots of our solution. Finally, the project management is presented containing a final project evaluation.
# Contents

## Acknowledgements

## Introduction

1. Project Description ........................................ 1
2. Project Goals ........................................... 2

## Research

2.1 Literature Review ...................................... 3
   2.1.1 Benefits of screen capture ....................... 3
   2.1.2 Capture of multimedia ............................. 3
   2.1.3 Note-taking and annotation of multimedia ....... 4

## Analysis

3.1 Problem Scenario ...................................... 7
3.2 Negotiation with Customer ............................. 7
3.3 Requirements .......................................... 8
   3.3.1 Functional ........................................ 8
   3.3.2 Non-functional ................................... 8

## Design

4.1 High-level System Overview ......................... 13
4.2 Hardware ............................................. 14
4.3 Application Programming Interface .................. 15
4.4 Server Architecture .................................. 18
   4.4.1 Application ....................................... 18
   4.4.2 Database ......................................... 18
4.5 User Interface ....................................... 20
   4.5.1 Device Specific Design ......................... 20
      4.5.1.1 Web Interface ............................... 20
      4.5.1.2 iPhone App ................................ 20
      4.5.1.3 Android Application ...................... 21
   4.5.2 Prototype Designs ............................... 21
      4.5.2.1 Note Taking Interface .................... 21
      4.5.2.2 Lecture Recording Interface .............. 23
## 5 Implementation

5.1 Low-level System Overview .............................................. 24  
5.2 Application Programming Interface ................................... 25  
5.3 Client Architecture ..................................................... 27  
5.4 Server Architecture ..................................................... 28  
5.4.1 Lecture Capture Event Flow ......................................... 28  
5.4.2 Capturing Streams ...................................................... 29  
5.4.2.1 Red5 ......................................................................... 30  
5.4.2.2 Applying OOP Principles ........................................... 30  
5.4.2.3 Database ............................................................... 30  
5.4.2.4 Device Control ....................................................... 32  
5.4.2.5 Downloading Lecture Multimedia ................................ 34  
5.4.2.6 Error Handling ....................................................... 35  
5.4.2.7 Maintaining State .................................................... 35  
5.5 Synote Integration ........................................................... 36  
5.6 User Interface ............................................................... 37  
5.6.1 Development .............................................................. 37  
5.6.1.1 Iteration 1: Prototype ................................................ 37  
5.6.1.2 Iteration 2: Styling and API ....................................... 37  
5.6.1.3 Iteration 3: JavaScript Integration .............................. 38  
5.6.2 Web Interface ............................................................ 38  
5.6.3 Mobile Interface ......................................................... 41

## 6 Testing

6.1 Overview ........................................................................... 44  
6.2 Unit Testing ................................................................. 45  
6.2.1 User Interface ............................................................ 45  
6.2.2 Media Capture .......................................................... 45  
6.2.3 Server Behaviour ....................................................... 46  
6.2.3.1 API ................................................................. 46  
6.2.3.2 Java ............................................................... 47  
6.3 Integration Testing .......................................................... 48  
6.3.1 User Interface and API ................................................ 48  
6.3.1.1 Applet and Flex Integration .................................... 48  
6.3.2 Java Server and Media Capture Integration .................... 49  
6.3.2.1 Screen Streaming .................................................. 49  
6.3.2.2 Webcam and Microphone Streaming ......................... 49  
6.3.3 API and Java ............................................................ 50  
6.4 System Testing ............................................................. 50  
6.5 User Acceptance Testing ................................................. 51  
6.6 Solution Evaluation ....................................................... 52

## 7 Project Management

7.1 Project Planning ............................................................ 53  
7.2 Software Lifecycle ......................................................... 55  
7.3 Risk Management .......................................................... 56  
7.3.1 Iteration 1 ............................................................... 56
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3.2 Iteration 2</td>
<td>56</td>
</tr>
<tr>
<td>7.3.3 Iteration 3</td>
<td>57</td>
</tr>
<tr>
<td>7.4 Source and Documentation Control</td>
<td>58</td>
</tr>
<tr>
<td>7.5 Group Work</td>
<td>58</td>
</tr>
<tr>
<td>7.6 Individual Work</td>
<td>61</td>
</tr>
<tr>
<td>8 Evaluation</td>
<td>63</td>
</tr>
<tr>
<td>8.1 Project Outcome</td>
<td>63</td>
</tr>
<tr>
<td>8.2 Achievements</td>
<td>63</td>
</tr>
<tr>
<td>8.3 Critical Evaluation</td>
<td>64</td>
</tr>
<tr>
<td>8.4 Future Work</td>
<td>64</td>
</tr>
<tr>
<td>8.4.1 Further Integration</td>
<td>64</td>
</tr>
<tr>
<td>8.4.2 Features</td>
<td>65</td>
</tr>
<tr>
<td>8.5 Final Remarks</td>
<td>66</td>
</tr>
<tr>
<td>Bibliography</td>
<td>67</td>
</tr>
<tr>
<td>A Risk Assessment</td>
<td></td>
</tr>
<tr>
<td>A.1 Initial Risk Assessment</td>
<td>1</td>
</tr>
<tr>
<td>A.2 Secondary Risk Assessment</td>
<td>2</td>
</tr>
<tr>
<td>A.3 Final Risk Assessment</td>
<td>2</td>
</tr>
<tr>
<td>B Server File Structure</td>
<td>3</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Problem Scenario .................................................. 1
1.2 Synote ................................................................. 2
2.1 Camtasia Studio ...................................................... 6
2.2 Screen Toaster ....................................................... 6
3.1 Use Case Diagram ................................................... 10
4.1 High level overview of the proposed system ...................... 13
4.2 System State diagram .............................................. 14
4.3 Conceptual Server Class Diagram ................................. 19
4.4 Database Entity Relationship Diagram .......................... 19
4.5 Navigation between pages of user interface ..................... 21
4.6 Log on screens for web and mobile interfaces .................. 22
4.7 Course selection screens for web and mobile interfaces ........ 22
4.8 Note taking screens for web and mobile interfaces ............ 23
4.9 Interface design for lecture capture ............................. 23
5.1 System Overview .................................................... 25
5.2 Code Extract from the API wrapper (index.php) ................. 26
5.3 Communication between Flash application and Java applet via JavaScript .............................................. 27
5.4 noteEd API Sequence ................................................. 28
5.5 Flex Sequence ........................................................ 29
5.6 Red5 Application Configuration ................................... 31
5.7 Observer Pattern [15] ............................................... 33
5.8 Starting a Stream Broadcast ....................................... 33
5.9 Singleton Pattern [13] ............................................... 33
5.10 Flowchart of screen capture post-processing .................... 36
5.11 Prototype Screenshots ............................................. 37
5.12 Log in and sign up screens ...................................... 38
5.13 Navigation screens to locate a lecture .......................... 39
5.14 Lecture screens throughout a lecture ........................... 40
5.15 Mobile log on screens ............................................. 41
5.16 Mobile courses screens ......................................... 42
5.17 Mobile lectures screens ......................................... 42
5.18 Mobile notes screens ............................................. 43
6.1 V-model Software Development Process ........................ 44
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Initial Gantt Chart</td>
<td>54</td>
</tr>
<tr>
<td>7.2</td>
<td>Secondary Gantt Chart</td>
<td>54</td>
</tr>
<tr>
<td>7.3</td>
<td>Final Gantt Chart</td>
<td>55</td>
</tr>
<tr>
<td>7.4</td>
<td>Boehm’s Spiral Model</td>
<td>55</td>
</tr>
<tr>
<td>7.5</td>
<td>Collaborative Timetable</td>
<td>59</td>
</tr>
<tr>
<td>7.6</td>
<td>Group hours worked per week</td>
<td>59</td>
</tr>
<tr>
<td>7.7</td>
<td>Source code commits and group hours worked per week</td>
<td>60</td>
</tr>
<tr>
<td>7.8</td>
<td>Breakdown of group hours worked</td>
<td>60</td>
</tr>
<tr>
<td>7.9</td>
<td>Individual hours worked per week</td>
<td>61</td>
</tr>
<tr>
<td>7.10</td>
<td>Breakdown of individual hours worked</td>
<td>62</td>
</tr>
<tr>
<td>B.1</td>
<td>Server File System</td>
<td>3</td>
</tr>
</tbody>
</table>
Acknowledgements

We would like to thank Mike Wald for supervising us throughout our project and providing us with his expert knowledge of the field. We also thank Tim Chown for his continued support and the Synote developers Yunjia Li and Priyanka Singh for their assistance in noteEd - Synote integration. Finally, the help of Andrew Landells for setting up and assisting with our development server was much appreciated.
Chapter 1

Introduction

1.1 Project Description

This project is concerned primarily with the capture of multimedia from lectures. A custom web-based application, noteEd, has been proposed for the purpose of recording a lecturer’s screen, microphone and webcam. The recordings will be streamed to an external server, and not saved on the local machine. A fourth form of media has been identified as the notes created by students during a lecture. These electronic notes could also be captured by the noteEd system. The problem scenario is illustrated in figure 1.1.

![Problem Scenario](image)

**Figure 1.1: Problem Scenario**

Streaming the captured media to an external server benefits the author of the recording in a number of ways. First, it enables the software to function independently from the local machine allowing lecturers to use any computer with a web browser to capture multimedia. Second, the
Chapter 1 Introduction

A lecturer does not need to manually convert or upload any multimedia after the lecture. Finally, timestamps can be recorded to aid the synchronised playback of all multimedia.

NoteEd aims to fill niche functionality of an existing application, Synote. Synote (figure 1.1) is a web-based application that synchronises the playback of various multimedia captured during lectures. Currently, Synote supports the synchronisation of video, textual transcripts, bookmarks and slides, all of which require accurate timestamps to ensure accurate synchronisation.

1.2 Project Goals

In this project we intend to complete the following goals. As the project leader of Synote, Dr Mike Wald will be our customer for the noteEd project. He intends to use our system to supplement Synote and complete the lecture playback package.

1. Capture multimedia from lectures
   (a) Screen capture
   (b) Microphone capture
   (c) Webcam capture

2. Allow students to upload lecture notes live

3. Provide an interface to Synote
Chapter 2

Research

2.1 Literature Review

2.1.1 Benefits of screen capture

Haga [11] discusses the benefits of using discussion boards with embedded video for distance learning, similar to the way Synote provides the video alongside the student’s notes. Notes are synchronised with the video by sending a relative timestamp with their message.

Abowd et al. describe the Classroom 2000 project [1], which is primarily concerned with automated capture, integration and access to lectures. The paper focuses on a system designed to capture both the use of an electronic whiteboard and the audio track. The project involved eighteen months of everyday use of the system and followed an iterative development style to assess the uptake of their system by students and lecturers. Usage analysis suggested that there was a strong demand for on-line access to lecture material, although most students did not feel it improved their performance in class.

2.1.2 Capture of multimedia

Gill [10] discusses techniques for capturing multimedia from the classroom. He outlines some simple guidelines for storing media, choosing formats and presenting. These aim to be practical tips for increasing the efficiency and utility of the final recordings. Gill observed that the quality of video recorded had little effect on the satisfaction of the students in the study. However he also found that the availability and use of captured multimedia did improve student performance.

Cruz and Hill describe techniques and considerations for the digital capture and playback of presentations and lectures [8]. The authors introduce a prototype system, STREAMS, to record audio and video streams in a searchable and accessible format. They discuss the key factors relating to media capture, including its obtrusive nature, media quality, completeness of material,
stream separation and cost. However, the STREAMS system makes use of extensive hardware installation, suggesting poor portability and tight coupling to its environment.

Abowd et al. present lessons learnt from the use of various initial prototypes created for the Classroom 2000 project [2]. The most significant insight is that of live prototype testing. The authors state that without the live demonstrations important student feedback would have been missed. Additionally, the inclusion of video data is essential to fully capture lecturer-student interaction. The paper concludes by mentioning students’ positive response toward the use of online note-taking.

The issues with the capture of lecture material are later discussed in another paper authored by Abowd et al. [1]. The authors describe the difficulty of providing sufficient human support at the start of the lecture. The desire to start recording multiple information streams required a heavy human involvement to ensure that all streams are synchronised. However, in future, the project aims is to enrich the audio and screen capture capability with the option to record video data as well.

A similar project, eClass, focusses on the automated capture and access of lecture materials [5]. Brotherton and Abowd describe the goal of eClass; to produce an environment where electronic notes, audio and video recordings can be captured during lectures and preserved for future access. With regard to material capture, the eClass system primarily highlighted the difficulty in creating a flexible screen capture tool. Specifically, the authors state that any format of presentation should be able to be captured with zero set-up time. Additionally, the poor quality and high bandwidth requirements of video was cited as a reason why video was not heavily used in their studies. Finally, the lack of collaborative note taking functionality was listed as a limitation of the eClass system. However, it was suggested that this functionality should be easier to implement with the introduction of wireless laptops to the classroom.

Mukhopadhyay and Smith describe a system to construct multimedia documents from live presentations, consisting of audio, video, image and textual data [14]. The paper focuses on two fundamental problems; synchronisation of captured data and automatic editing. The synchronisation issue arises when captured material has no synchronised timestamp information associated with it, and is therefore not applicable to this project. However, an algorithm is suggested to automatically edit multiple video streams together in to a single stream. This therefore requires a heavy amount of post-lecture video analysis, editing and encoding.

2.1.3 Note-taking and annotation of multimedia

Wald et al. state the power of annotating a specific section of a multimedia recording [16]. The authors mention the need to collate notes created by more than one user in addition to requirements relating to lecture transcripts. A system named Synote is presented, which provides functionality for playback of a range of synchronised media. Students’ notes can act as video annotations, easing tasks such as access, search and management of lecture material.
The power of asynchronous video annotation is discussed by Bargeron et al. using the MRAS system [3]. MRAS is an electronic video annotation system which could be compared to hand written notes. The conclusions point toward a generally positive response to shared online note-taking. Cited advantages over traditional note-taking include enriching on others’ notes, aiding preparation for class discussion and increasing the volume of notes possible through pause and rewind of material.

Chiu et al. [6] present NoteLook; a system to digitally simulate video and hand-written notes produced during meetings. The system consists of a number of video cameras and stylus-based computers used to capture multimedia from meetings. The authors state a number of advantages to capturing this material, including being able to catch up on meetings and rich indexing possible to aid searching. Implementation challenges are discussed, with emphasis on building a lightweight system that does not hinder the meeting attendees. Further mentioned are the drawbacks of using substantial infrastructure, such as previously set up video cameras and expensive pen-based laptops. The final consideration was the decision to automatically capture slides to reduce the complexity of note-taking.

2.2 Review of Existing Applications

There are a number of existing applications for screen capture. Camtasia make the most well known desktop solution; Camtasia Studio. This provides many features including mouse pointer highlighting, selection of recording area and video editing. There are also some free applications such as CamStudio for Windows, which is available as open source software.

Camtasia also produce an application called Camtasia Relay which is aimed at presentation capture. This uses a client-server architecture to offload the processing from the capture client onto a central server. However this still requires the client-side PC to have the capture application installed. Fig 2.1 shows a screenshot of the Camtasia studio application.

Web based screen capture is a much more recent development. However, due to the rapid development environment of the web, there are already a few applications available. Many of these provide similar functionality, however, all current solutions are limited in a way that makes them completely unusable for lecture capture.

Screencast-o-Matic (http://www.screencast-o-matic.com) is possibly the original. Is is an entirely Java based application which allows the user to select an area of the screen to record, and captures microphone input. It provides an export function to convert the video to Quicktime (MOV), Windows Media Video (WMV) or Flash Video (FLV) formats. One of its limitations is that it has a maximum recording duration is 15 minutes. However, Screencast-o-Matic license their capture code to others, so there are a few similar applications which utilise their code.

Screen toaster (http://www.screentoaster.com) uses a combination of Flash and Java to deliver their application. Due to the use of Flash, their application can offer webcam capture in addition
to desktop and audio capture. It also provides the ability to playback screen capture at a speed faster or slower than realtime. Screen toaster has, instead of a time limitation, a filesize limit of 20 MB on the media which can be captured. As all video is stored on the client machine, it must therefore be uploaded after the recording has finished. Our application avoids this situation by sending the data during the capture process, so there is not a backlog of image data to send after the capture. Fig 2.2 shows the website for the screen toaster tool.
Chapter 3

Analysis

3.1 Problem Scenario

Synote is an application developed by Dr Mike Wald and the Synote development team within the Learning Societies Lab research group at the University of Southampton. Synote is a web-based application that synchronises the playback of various multimedia captured during lectures. However, Synote offers no functionality to capture any multimedia. Although various products exist that can record specific devices and have good compatibility with video hosting websites, the functionality to download the media later does not exist. This requires lecturers to either download the files or upload them to a video hosting site instantly upon capturing the media.

It is also common for students to take notes during lectures. These notes can be hand-written, typed or drawn on a tablet using a stylus. A wide range of collaborative note taking tools exist, but integrating such tools into a multimedia capture system would be beyond the scope of this project. Additionally, collaborative note taking relies heavily on accurate, synchronised timestamps, as the topic of speech can change rapidly in a lecture.

3.2 Negotiation with Customer

Negotiation with the customer took place throughout the requirements elicitation process. Feasible components were discussed with the aid of prototypes, while research was carried out on the more challenging features. The team’s widely distributed skill set was fully exploited by negotiating the requirements in order to produce the most appropriate solution.

One heavily discussed topic was that of the note taking tool. It was suggested to take advantage of existing systems such as Twitter or Microsoft OneNote. However, technical issues such as the lack of timing information and the possible note posting latency created the need for
an additional system. The decision was reached to create a note-taking Application Programming Interface (API) and a number of user interfaces to solve the problem. This produced the requirements list stated in the next section.

3.3 Requirements

The following requirements capture the core functionality expected of the system on completion. With the aid of the customer, these requirements will be used to evaluate the produced system. The requirements have been broken down into functional and non-functional requirements; depending on whether they directly affect the system’s functionality or act as a system constraint. Should the core requirements be completed ahead of schedule, a number of functional extensions have been listed in order of priority.

3.3.1 Functional

1. Lecturers must be able to record multimedia lecture materials from the following devices:
   
   (a) Microphone
   
   (b) Webcam
   
   (c) Screen capture

2. All multimedia must be time stamped with synchronised creation times

3. Students must be able to create lecture notes during lectures

4. Lecturers must be able to download captured multimedia

5. All multimedia must be available in a form compatible with Synote

Extensions

1. An API must be provided to access the lecture multimedia

2. Lecturers must be able to review captured multimedia online

3.3.2 Non-functional

1. A web-based user interface must be available for:
   
   (a) Lecturers to record multimedia
   
   (b) Students to submit lecture notes
Chapter 3 Analysis

2. Lecture videos must be available in a number of formats, including Windows Media Video (WMV) and Flash video (FLV) file formats

3. All multimedia must be stored initially on a central server

4. Reasonable security must be employed to protect recorded material from deliberate and accidental damage

5. Sufficient documentation must be provided with the solution

6. A standardised software engineering approach must be followed throughout the project

3.4 Use Case Analysis

Use Case Analysis was carried out to describe the system’s behaviour and its interaction with users. Figure 3.1 visually represents the functional requirements of the system. It was decided to separate the functionality available to the lecturer and students. Both students and lecturers would be able to login to the system and browse to a specific lecture. However when a lecture is selected, the lecturer would be provided with the ability to capture multimedia, whereas students would only be able to take notes.

3.4.1 Analysis of Technology

In order to allow other software to interact with the noteEd system it will necessary to provide an Application Programming Interface. By opening up the functionality of the noteEd system it eases the creation of a number of interfaces. The use of such an API abstracts the vast complexity of the system away from its interfaces, reducing code duplication among each interface. Implementing a Representational State Transfer (RESTful) architectural style would ensure the scalability and generality of the interface, and would be particularly suited to platforms with reduced libraries or strict hardware limitations. Last.fm is an online music service which implements such an API. The Last.fm API responds in Extensible Markup Language (XML) to methods expressed as arguments passed to the root URL. Technologies well suited to the implementation of such an API include PHP and JSP.

When interacting with our API in the web-browser, it is preferable to provide instant responses to any requests. This promotes a live look and feel to the system. A comparison of push and pull techniques for Ajax [4] details a number of methods used to transfer data persistently between a server and client. Bozdag, Mesbah and Deursen explain the concept of an Ajax XMLHttpRequest to make use of RESTful services, before explaining how persistence can be added through the inclusion of a time to refresh (forcing the page to re-load every time period).

When a web page needs to respond to certain events invoked on the server, a technique known as Reverse AJAX or Comet is used. Comet enables the server to send messages to the client when
an event occurs, without the client having to send a request explicitly. The paper proceeded to
analyse the performance of these methods. It should be noted that all the described techniques
scale well, although the paper hastens to point out that naive implementations would be likely
to negatively impact on performance. To aid the development of such services, Comet and
Reverse Ajax: The Next Generation Ajax 2.0 [7] suggests several practical techniques such as
intelligently altering the time to refresh of an HTTP Pull, based on the rate of previously received
new data.

To facilitate streaming of media, there are a number of cross-browser, cross-platform plug-
ins available. These are Java, Flash and Silverlight. Silverlight is a younger technology, and
therefore is yet to reach the mass coverage of Flash or Java. Silverlight allows for scripts to
be written using C# and Visual Basic.Net, as well as JavaScript on the client side. However
it is known that some lecturers use Linux, an operating system for which Silverlight is not yet supported, whereas Java and Flash are.

Video support is very important for the purpose of the noteEd system, so it is essential that the plugin used to provide support for standard codecs. In terms of support for capture devices such as microphones and webcams, the Flash environment provides very simple access to any device that the operating system provides drivers to. In contrast, the use of Java would require substantial initial development time, needed for constructing a system to access these devices in a consistent and reliable manner.

When developing for Flash, the option is available to use Adobe Flex; a framework for building rich web applications. It uses a combination of an XML based interface specification language and the ECMAScript based ActionScript, which define the interface and behaviour of an application respectively. The application can then be compiled to run on the Flash plugin within the browser.

Additionally, Adobe sell Flash Media Server, a family of products designed for streaming video and real-time communication. However, the product’s licence fee is well out of the range of this project’s budget. An alternative technology is the open source flash server, Red5. Red5 is a Java implementation of a flash media server, allowing the recording of client media streams. Alternatively, if both the client and server were to be developed using Java, communication and media streaming between the two would be significantly simplified. However this may be at the expense of a larger applet, requiring a longer initial download for the user.

Whilst there were a variety of options for streaming the webcam and microphone, Flash and Silverlight could not provide a solution to capture the user’s desktop due to their security models. The only technology we had found which can capture the user’s desktop is a Java applet. However, applets in Java also have a security sandbox, which requires the applet to be digitally signed. The user therefore have to accept a security certificate before it can access the desktop.

To use both Adobe Flex and a Java applet would either require user interaction with two interfaces or some communication between the two technologies. Merapi is a bridge between Adobe AIR applications and Java which could be used to trigger the start of all device streams. Another method of communication is available through the web page’s JavaScript. Both Flex and Java applications running inside a webpage have access to the page’s JavaScript, therefore allowing communication such as ‘Start Capture’ to be sent between the applications.

As the Java applet only captures images of the lecturer’s screen, they are required to be compiled into a video file. Java includes the the Java Media Framework in the standard library, which can be used to create a Quicktime video file from a stream of JPEG images. However, the noteEd system is required to support more video formats than just uncompressed Quicktime, which is very inefficient in its use of disk space and bandwidth.

FFmpeg is an open source library tool designed to record, convert and stream audio and video. Additionally, there are many libraries that provide easy access to its functionality. Xuggle is
one such open source library that can manipulate and compress live video. The Xuggle library supports many formats and provides a robust way to create video from images. This would be ideal for usage on the server.

It is likely the noteEd system will make use of a back-end database to persistently store the system’s data. The use of a mySQL database would allow a number of applications, written in a variety of languages, to query and update the database. It is expected that the API for the noteEd system will make use of database connections such as Java’s JDBC and PHP’s ODBC functions.

### 3.4.2 Analysis of Resources

The nature of this project requires the proposed system to process and transmit large volumes of data. Although high-performance hardware would be necessary should the system be deployed for practical use, it is not needed during its development. Therefore, a remote server has been acquired for development and testing purposes. It has the following specification:

- Processor: Intel Xeon CPU @ 2.13 GHz
- Memory: 1.00 GB
- Storage: 126 GB

The transmission of large volumes of data across the Internet also requires a high bandwidth Internet connection. A speed test was completed to assess the quality of the Internet connection. It was concluded that the download speed of the server was in excess of 60 MB/sec and upload speed in excess of 15 MB/sec. These speeds are more than adequate for development purposes.

As discussed during the technology analysis, Adobe’s Flash Media Server provides a feature rich but expensive solution to media stream recording. However, this project doesn’t have the licences for such software or the budget to buy one. However, Adobe do provide student licenses for their IDE, Flex Builder, which we acquired and will make full use of. Flex Builder is built on top of the open source IDE Eclipse, which all the group has experience with. Flex Builder provides an integrated environment for development with the Adobe Flex SDK.

Through analysis we found many of the tools we needed were out our budget but we managed to find enough open source software to start the project. These tools provide functionality equivalent to commercial software, although we should expect some common issues with open source, such as lack of documentation or product support.
Chapter 4

Design

4.1 High-level System Overview

The noteEd system aims to provide functionality to its users through a range of devices. Figure 4.1 shows the full list of devices that interfaces will be designed for. In this diagram, the server is treated as a black box, and so focuses on the interaction between each device and the server.

![Figure 4.1: High level overview of the proposed system](image)

For note taking, the noteEd system has interfaces for both a student’s computer and mobile device’s web browsers. On mobile devices, interfaces have been designed as native applications for both the iPhone and Android powered devices. Through the use of these, the noteEd system aims to make note taking functionality as accessible as possible.

The state diagram in figure 4.2 describes the actions the user can take when logged in to the interface. It highlights the ability to either move through courses to lectures or skip straight to
important lectures. There is also the need to log off at any point which suggests the use of a navigation bar where relevant links and information can be stored.

![System State Diagram](image)

**Figure 4.2: System State diagram**

### 4.2 Hardware

Many current lecture capture systems require specialised recording hardware to be installed in the lecture theatre. However, this equipment is often expensive and not easily transported. Our decision to provide an interface for capture, along with standards based clients, means that any hardware recognised by the operating system can be used for capturing multimedia. This allows lecturers to use whichever recording equipment they are comfortable with, as well as reducing cost by allowing re-use of existing hardware. The Flex interface will allow the user to select which microphone or camera to use, adding flexibility and choice for the lecturer.

Additionally, due to the nature of our web based system, the set up and shut down time is kept to a minimum. The applet and flash code will be cached, further improving start up time. All post-capture processing will be carried out automatically on the central server, allowing the media to be downloaded by the lecturer as soon as possible.
4.3 Application Programming Interface

As noteEd was required to be accessible on a plethora of systems over the web, we decided it was sensible to create a RESTful API (where requests could be passed over the GET path). This meant that we could tailor the output of our API to increase device performance and allowed all parties efficient access of core functionality in noteEd, while protecting our system via a layer of abstraction.

With a diverse range of operations required from the API it was important to create a generic framework where methods could be added with ease. Therefore, an object oriented approach seemed ideal. By utilising polymorphism, we could treat all methods on the API as equals, simply running the appropriate class when required.

The API we designed was based around the notion of an apiMethod. An apiMethod is an abstract class, which every method on the API must extend. Methods on the API are required to define a responder (the method argument to which they respond) and a process which they perform, along with relevant documentation. A wrapper script then scans the methods directory and instantiates the methods, storing them associatively by their responders. Methods can then be called by name and the response converted into either JSON (JavaScript Object Notation) or XML.

The solution we designed had many benefits. By using a drop-in approach to API development, our system could be easily extended by creating new classes. This atomic approach ensured that the API was not broken by having to edit any configuration files. The highly modular API meant that errors could be easily trapped and located. By creating generic method definitions, methods could be developed without restricting functionality. An important design decision was made during this process, that SHA-1 hashes would be passed between the client and server as OpenIDs to uniquely identify a user’s session. These session hashes would be generated from an authentication API method, which also required a SHA-1 hashed password as input.

As the API would most frequently be read in XML format, we decided that the JSON would be generated from the XML using IBM’s xml2json [12]. This allowed the design to remain clean, and didn’t unnecessarily bloat the size of individual methods. Before implementation could begin, we had to ascertain the full functionality of our system. The listing of API functions is provided in figures 4.1 and 4.2. This design stage would allow us to create an API which granted secure and efficient access to our system through a standardised form.

With the potential of storing a large amount of information in the database, we needed to ensure that the data was stored in the most efficient manner. Originally, we decided to use the MyISAM storage engine as this would drastically reduce the size of our database. However, on closer inspection we realised that our system would greatly benefit from transactional support and the size would be smaller than we initially anticipated. Therefore, we decided to switch to InnoDB.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth</td>
<td>Returns the hash key stored in the session table</td>
<td>‘user’ - username</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘pass’ - password hash</td>
</tr>
<tr>
<td>createLecture</td>
<td>Creates a given lecture. Requires user to be a lecturer</td>
<td>‘hash’ - hash returned from auth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘startingAt’ - (optional) timestamp for the start of the lecture</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘withCourseCode’ - course code for the lecture</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘withLecturer’ - (optional) lecturer giving the lecture</td>
</tr>
<tr>
<td>createUser</td>
<td>Creates a user without lecturer rights</td>
<td>‘withUserName’ - username</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘withPassword’ - password hash</td>
</tr>
<tr>
<td>disableUser</td>
<td>Disables the user from the system</td>
<td>‘hash’ - hash of the user, returned from auth</td>
</tr>
<tr>
<td>getCourses</td>
<td>Returns all the courses stored in the database</td>
<td>‘hash’ - hash returned from auth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘limitedTo’ - (optional) LIMIT on number of courses</td>
</tr>
<tr>
<td>getDevices</td>
<td>Returns a list of names and IDs of available devices</td>
<td>‘hash’ - hash returned from auth</td>
</tr>
<tr>
<td>getFiles</td>
<td>Returns directory structure of all content of a given lecture</td>
<td>‘hash’ - hash returned from auth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘withCourseCode’ - the courseCode of the lecture</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘withLectureNo’ - the lecture number of the lecture</td>
</tr>
<tr>
<td>getLectureDetails</td>
<td>Returns the details of a lecture</td>
<td>‘hash’ - hash returned from auth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘withCourseCode’ - the courseCode of the lecture</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘withLectureNo’ - the lecture number of the lecture</td>
</tr>
<tr>
<td>getLectureDownload</td>
<td>Returns a zip file containing all files generated for the lecture</td>
<td>‘hash’ - hash returned from auth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘withCourseCode’ - the courseCode of the lecture</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘withLectureNo’ - the lecture number of the lecture</td>
</tr>
<tr>
<td>getLectureNotes</td>
<td>Returns all the notes for a given lecture</td>
<td>‘hash’ - hash returned from auth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘withCourseCode’ - the courseCode of the lecture</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘withLectureNo’ - the lecture number of the lecture</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘limitedTo’ - (optional) LIMIT on number of lectures</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘withVisibility’ - (optional) either ‘self’ or ‘public’</td>
</tr>
<tr>
<td>getLectures</td>
<td>Returns all the lectures stored in the database</td>
<td>‘hash’ - hash returned from auth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘limitedTo’ - (optional) LIMIT on number of lectures</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘withCourseCode’ - (optional) the courseCode of the lectures</td>
</tr>
<tr>
<td>getLecturesInProgress</td>
<td>Returns the lectures currently in progress stored in the database</td>
<td>‘hash’ - hash returned from auth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘limitedTo’ - (optional) LIMIT on number of lectures in progress</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘withCourseCode’ - (optional) the courseCode of the lectures</td>
</tr>
</tbody>
</table>

**Table 4.1: API Methods: Section 1**
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>getNow</td>
<td>Returns the server timestamp to the nearest second in ISO8601 format</td>
<td>‘hash’ - hash returned from auth</td>
</tr>
<tr>
<td>getServerConfig</td>
<td>Server responds with its name and a port assigned for the connection</td>
<td>‘hash’ - hash returned from auth</td>
</tr>
<tr>
<td>getUpcomingLectures</td>
<td>Returns the upcoming lectures stored in the database</td>
<td>‘hash’ - hash returned from auth, ‘limitedTo’ - (optional) LIMIT on number of upcoming lectures, ‘withCourseCode’ - (optional) the courseCode of the lectures</td>
</tr>
<tr>
<td>makeLecturer</td>
<td>Makes a user account into a lecturer account</td>
<td>‘hash’ - hash returned from auth, ‘withAuthorisation’ - the authorisation code required to upgrade</td>
</tr>
<tr>
<td>startLecture</td>
<td>Sets the status of a lecture to ‘In Progress’</td>
<td>‘hash’ - hash returned from auth, ‘withCourseCode’ - the courseCode of the lecture, ‘withLectureNo’ - the lecture number of the lecture</td>
</tr>
<tr>
<td>startRecording</td>
<td>User requests the recording to commence</td>
<td>‘hash’ - hash returned from auth, ‘withCourseCode’ - the courseCode of the lecture, ‘withLectureNo’ - the lecture number of the lecture, ‘withDeviceID’ - identification number of device as from listDevices</td>
</tr>
<tr>
<td>stopLecture</td>
<td>Sets a given lecture to Finished</td>
<td>‘hash’ - hash returned from auth, ‘withCourseCode’ - the courseCode of the lecture, ‘withLectureNo’ - the lecture number of the lecture</td>
</tr>
<tr>
<td>stopRecording</td>
<td>User requests the recording to end</td>
<td>‘hash’ - hash returned from auth, ‘withCourseCode’ - the courseCode of the lecture, ‘withLectureNo’ - the lecture number of the lecture, ‘withDeviceID’ - identification number of device as from listDevices</td>
</tr>
<tr>
<td>submitNote</td>
<td>Submits a note to a given lecture</td>
<td>‘hash’ - hash returned from auth, ‘with CourseCode’ - the courseCode of the lecture, ‘withLectureNo’ - the lecture number of the lecture, ‘withMessage’ - note contents, ‘startedAt’ - timestamp at which note was started</td>
</tr>
</tbody>
</table>

**Table 4.2: API Methods: Section 2**
4.4 Server Architecture

4.4.1 Application

The server is a critical part of the system and has a variety of functions to provide. In order to make debugging and maintenance as simple as possible, the breakdown of functionality needed to be carefully considered, along with a sensible approach to accessing system constants and variables. To this end, as little configurable data should be embedded in code as possible. Instead, files and the database should be used to store values used by the various classes. Therefore, there will be a class to handle file system related tasks and a class to handle device related tasks (a device in the server is the wrapper around a stream, allowing it to be read). There will be a factory to instantiate devices with utilisation of inheritance and polymorphism to generalise device objects. The use of object orientated principles means that Java will be a good choice of language for this system.

Other functions of the server are the single-point of connection to the PHP API and a single Red5 application to interact with the Red5 framework. A housekeeping utility is required to keep the file system and database clean and tidy. There will be utility classes which perform common general-purpose tasks such as connecting to the database. Relevant exceptions will be used for caught errors but only one exception type will be thrown to the API. There are many external Java libraries to simplify tasks such as scheduling, stream management and database querying. A graphical view of these requirements is shown in figure 4.3. It highlights the points of entry to the server and the internal classes.

4.4.2 Database

An SQL database will provide a permanent storage system which can hold configuration data as well as user generated content. Figure 4.4 describes the database tables and their interaction. A lecture is identified by a combination of a course code and a lecture number; this pair is used when querying other tables and is included in many API functions. Any API functions that do not require Java functionality will access the database straight from the PHP.
Figure 4.3: Conceptual Server Class Diagram

Figure 4.4: Database Entity Relationship Diagram
4.5 User Interface

Interaction between devices and our system is facilitated by the noteEd API. Several design decisions about the API dictated the ways in which it could be used. The implementations we designed made best use of the API and are recommended for any future extensions to our projecting using the API.

4.5.1 Device Specific Design

4.5.1.1 Web Interface

The primary point of contact for a user on our system is the web interface. We wanted our web interface to provide a friendly, modern experience for users, with seamless transitions between the pages. As we designed the API to include a JSON output, it was sensible to implement a Web 2.0 style interface utilising AJAX and JavaScript’s JSON parsing.

With a focus on clean interaction to the API and good coding practices, we created a Prototype (JavaScript Object) to deal with the processing of API requests and responses. This allowed us to easily design an interface for the site by partitioning the logic and styling and therefore following the MVC paradigm. By creating a Prototype, persistent information could be shared between the methods of the API, such as the session hash. With all the functionality contained in a single JavaScript object, the web interface’s source code remained clutter free and easy to edit.

4.5.1.2 iPhone App

The API for made it is possible to create interfaces for many devices. One device-specific interface we decided to implement was for the iPhone. iPhone Apps are written in Objective-C using the Cocoa Framework. This framework heavily uses MVC and follows a singleton design pattern where an AppDelegate is a static class. When integrating our API into such a framework, it was decided to place persistent data into the AppDelegate (singleton class). This meant it could be referenced by all other subsequent classes.

Within the iPhone App, the application was to follow a simple data flow where users would login and be presented with the choice to view current lectures, all courses, or information about the project. After selecting a lecture, users would be able view existing lecture notes and add new notes to the system. Each interaction, such as authentication, viewing lecture details or making a new note would require API interaction using the data functions of Objective-C, and the C-based libXML libraries. As the API requires SHA-1 hashes to authenticate a user, a cryptographic method was required to generate suitable parameters for the API.
4.5.1.3 Android Application

Another native application we decided to implement was an Android application. Android applications are written in the Java programming language and compiled into a package which can be deployed to an emulator or physical device. An Android application’s user interface is most commonly defined using XML, although it is also possible to define a view within Java code. This isolates screen presentation from business logic, providing a firm basis for following an MVC architectural pattern.

The Android application follows an identical data flow to the iPhone application; the user is initially presented with a login screen, after which the navigation menu can be accessed. Since both applications’ interaction with the noteEd API is very similar we decided to provide the same functionality. The XML returned by the API calls is interpreted in a similar way to the web interface, ensuring the only differences between the interfaces is due to device specific behaviour.

4.5.2 Prototype Designs

A number of user interface screens were designed for both web browsers and mobile devices. The mobile devices would need similar pages except the start, stop and download lecture facilities. Figure 4.5 shows the pages that would be created for the web interface. Links between pages are shown as arrows in the diagram. Additional links may be added to allow quick access to lectures for example lectures that are in progress.

![Figure 4.5: Navigation between pages of user interface](image)

It was decided not to create designs specific to a mobile device in an effort to create comparable native applications and mobile web interfaces. The designs for the log on screen are shown by figure 4.6. These designs are similar to each other, as the username, password and submit button components are present on both designs. The main difference between the web and mobile interface is the decreased level of detail in the mobile background.

4.5.2.1 Note Taking Interface

Figure 4.7 shows the web and mobile interface designs for the courses and lectures screens. These designs consist of a description section at the top of each screen, a list component and a
navigation section. However, the navigation section is of a different appearance and position in each design. In the web browser, it appears vertically in the left pane and in the mobile interface it appears horizontally at the bottom of the screen. It is intended that this navigation pane should contain links to lectures that are either in progress or upcoming, as well as links to the main courses list and an about page.

The designs for the web and mobile note taking screen are shown by figure 4.8. Both interfaces consist of an information section at the top, components to enter and submit a note, a list to view other notes and a navigation pane. The note taking component consists of a multiline text box and a submit button, while the navigation pane is identical to that of the previously described courses screen.
4.5.2.2 Lecture Recording Interface

The lecture recording interface will be a part of the main web interface. When lecturers log in they will have different privileges than the normal user. Firstly, they will be able to create new lectures and then start new recordings on lectures they have created. The recording interface is made up of a Java applet and a Flex interface. Figure 4.9 shows the design for the lecture capture interface. There will be options to select the media to capture and the devices to record from. Feedback will come in the form of web cam and microphone playback, which is easily done in Flex. This will help the lecturer to know that the capture devices have been set up correctly.
Chapter 5

Implementation

5.1 Low-level System Overview

Figure 5.1 shows the components of the proposed system. The system hosted on the server consists of an SQL database back-end, a Java application running inside an Apache Tomcat server and a PHP API. The users interact with the system through a web browser or a native application. These interfaces will be hosted on either a PC or a mobile device, such as a mobile phone.

An API was created in an effort to abstract complexity away from the user interface. This took the form of a RESTful API, written in PHP and hosted on the server. This API has been designed to respond to HTTP GET methods with an XML page, containing information about the status and result of the request. Web interfaces and native applications are therefore only required to send HTTP requests to the server and interpret the XML response to form the interface.

The PHP interface must also communicate with the Java application on the server. To facilitate this, a third party module called the PHP/Java Bridge was used. This bridge was used to call Java functions from PHP, such as functions to manage the file system when lectures are created or deleted and functions to serve a ZIP file containing all of a lecture’s multimedia. Both the PHP API and Java application were required to query and update the SQL database, using ODBC and JDBC respectively.

The designed system captures lecture multimedia using purely technology available inside the web browser. Flash will be used to capture and stream both the webcam video and microphone audio to the server. A Java applet is used to take screenshots of the users screen, which is streamed to the server. These media streams are saved on to the server using two third party Java modules, Red5 and Xuggle. Red5 is a Java implementation of a flash media server, allowing the webcam and microphone streams to be saved on the server. Xuggle is a Java wrapper for FFMPEG which can compile a live stream of images into a video, which will constitute the screen recording.
5.2 Application Programming Interface

From the outset of our project, we wanted to provide an interface that we and third parties could use to extend the system. To achieve this, we decided it was best to provide a RESTful API, outputting in JSON (for the web) and XML (for other applications). This required us to write in a server-side scripting language such as JSP or PHP.

Having decided to use Flex in the client to send video and audio streams, there was a requirement for our server to provide a Flash Media Streaming (FMS) service. As our system was to be provided for free, we were obliged to use a free FMS. We chose to use Red5, an open source flash media server - written in Java - to receive the streams from our clients. With the server now having a Java component, it would have seemed sensible to design the system using JSP and Java. However, after much consideration we realised this was not the best solution for our project.
When designing the system, we realised that our API would be highly database driven. Despite the performance increase of using compiled JSP, the ODBC MySQL drivers (used in PHP) are significantly faster than Java’s JDBC resulting in PHP actually out-performing JSP for an API like ours. Another key area of consideration was the bridge between Java and the API. If we had used Java and JSP, we would have simply have called the Java directly, in PHP however we had to use an extra component as the standard solution has been developed separately to the PHP language in recent versions. This is the PHP/Java Bridge and it works by replicating the Java Messaging Service (JMS) in PHP. This solution while not as efficient as JMS showed positive results, and ensured we had a hassle free method of communication between PHP and Java. In our system, the vast majority of the methods in the API rely on database access without needing to access the Java. With a large amount of experience designing RESTful API’s in PHP, we decided that it was best to create the API using PHP, as we could develop a more optimal solution at a faster rate without any loss of performance.

To provide the API the PHP scans the methods folder for apiMethods, associating their responders with the instantiated methods. The PHP uses the GET parameters in the URL to pick the appropriate method in the methods array and invokes its run() method. The result of this operation would be formatted in either JSON or XML as requested. This ensured the framework remained highly adaptable and easy to maintain, as plugins remained atomic yet allowed limitless variation. The code extract in Figure 5.2 (stripped of debugging methods to preserve readability) demonstrates the novel approach used by our API fully utilising the loose-typing of PHP.

```php
$definedClasses = get_declared_classes(); // Already Defined Classes
include_directory(METHODS_DIRECTORY); // Recursive Include Function
$activeMethod = array();
foreach (get_declared_classes() as $class) {
    if (!in_array($class, $definedClasses)) {
        $c = new $class();
        $activeMethod[$c->getResponder()] = $c;
    }
}                 
if (array_key_exists(METHOD, $activeMethod)) {
    $aM = $activeMethod[METHOD];
    if(strtolower($_GET['format']) == 'json') {
        header('Content-Type: text/javascript');
        header('Cache-Control: no-cache');
        echo xml2json::transformXmlStringToJson($aM->run());
    } else {
        header('content-type: text/xml');
        echo $aM->run();
    }
}
```

Figure 5.2: Code Extract from the API wrapper (index.php)
5.3 Client Architecture

The client browser streams media to the server via the Flex and Java interfaces. Java is used to capture sequential screenshots of the user’s desktop. The resolution of the transmitted screenshots is of excellent quality when the bandwidth is available. The Flex application is responsible for streaming the feeds from the lecturer’s microphone and/or webcam depending on the lecturers choice. The quality of each stream is configurable within Flex code. The webcam stream has been set to a resolution of 640×480 pixels and a frame rate of 10 frames per second.

During our first iteration we were developing using the Merapi Bridge to communicate between the Flash plugin and the Java plugin embedded in the webpage. However, use of Merapi required several libraries to be included in both the Flash and Java. This added download time for the client, and also created additional unwanted dependencies. Eventually we discovered we needed a new solution as we had issues getting the bridge to work when our application was deployed to the server.

We believe that these issues were due to security restrictions encountered due to being within a web browser. We found that the Merapi Bridge was not that well suited to being used in a totally web-based application. Instead, we moved to a custom JavaScript communication mechanism, whereby the Flash and Java make use of their ability to call JavaScript functions embedded in their web page. This turned out to be a much simpler mechanism and did not require too much additional development time.

Figure 5.3 describes the interaction between the Flash and the Java Applet on the Web interface. First the Flash verifies itself with the applet. After this it can send a message to start recording when the user presses the start button. When the lecture is over and the lecturer clicks stop, the Flash sends a final message causing the stop lecture function to be called.

Figure 5.3: Communication between Flash application and Java applet via JavaScript
5.4 Server Architecture

5.4.1 Lecture Capture Event Flow

As the noteEd server is API-driven, it serves XML as a feedback mechanism for the result of an API call. In order to stream multimedia to the server, the API calls must be made in a specific order as shown by figure 5.4, or else streaming will fail. The diagram assumes the user is logged in as a lecturer and has navigated to the lecture they wish to record.

![Diagram showing lecture capture event flow]

**Figure 5.4: noteEd API Sequence**

In Flash, the standard communication model to stream data from client to server is shown in figure 5.5. The Flash Media Server used is Red5 as it is open source, free, and written in Java, which simplifies integration with the server architecture. This model needs to be integrated with the noteEd client-server communication model to ensure correctness of usage.

In terms of sequence, the call to start a lecture can come before or after Flex connects. The start recording calls must come before the stream(s) are published, with the stop recording calls prior to the publishing being stopped. The Flex connection can be closed before or after the stop lecture call. Flex should publish stream(s) using the “live” mode, rather than the “record” mode to stop the Red5 framework from recording the stream automatically into a default folder with a default name. The noteEd server names and stores files in a specific location for the lecture.
5.4.2 Capturing Streams

Implementation of the noteEd server was complex as it had to accommodate multi-threading for simultaneous users, Red5 applications for stream management and noteEd API functions. To build a general-purpose system, i.e. one that could handle different types of stream delivered using different network protocols, configuration data was stored in files (for simple key value pairs) or the database (relational for data with more than one value per key).

The underlying interface is IDevice. This defines the methods a device (the endpoint for a stream) must implement in order to receive a stream in the server. A default abstract implementation can be extended for each type of stream the server is required to handle. The most interesting methods are prepare, read and tidy which are performed before, during and after a stream is received, respectively. The prepare and tidy methods map to API calls start recording and stop recording (although there is more to starting and stopping than these methods).

ImageDevice is used for screen capture; its read method uses Java sockets to receive images and feeds them into Xuggle which creates a video on the fly, encoded in whatever format specified. AudioDevice, VideoDevice and AudioVideoDevice interact with the Red5 framework to retrieve the stream being sent from a Flex client. This means a read method is rather simple, where the main two lines are:
Chapter 5 Implementation

```java
Cbs = (ClientBroadcastStream)
getApplication().getBroadcastStream(getScope(), getStreamName());
cbs.saveAs(filename, false);
```

The parameters and variables are all defined as part of the preparation stage. The tidying stage
stops recording and removes the entry in a map which identifies this stream as active.

5.4.2.1 Red5

The Red5 framework has default concrete implementations of its interfaces built in. By ex-
tending these, or implementing particular interfaces directly, it is possible to have complete
control over every aspect of the recording process. In order to save the stream to a file, with a
useful name in a suitable location, a class which implements the `org.red5.server.api.
stream.IStreamFilenameGenerator` interface will be used to handle the path of the
saved stream.

Red5 relies on the Spring framework and uses Java Beans, which is declared in the XML config-
uration of each application (see Figure 5.6 for the noteEd application). This tells it what classes
do what, so by pointing the relevant bean at our own `FilenameGenerator` class ensures
Red5 uses our method implementation and not its own. The XML is processed when Tomcat
is first started. Then, whenever `saveAs(…)` is called, the Red5 framework uses the Bean
settings to resolve the class to use for filename generation.

5.4.2.2 Applying OOP Principles

As before mentioned, the `Device` classes need to be instantiated in order to be used. A
`DeviceFactory` class was created to create instances of devices and populate the fields. In
order to generalise as much as possible, standard values for each class, including its name, were
stored in the database and retrieved when required. Then, using reflection, each class could be
instantiated and added to a list, which is then returned.

By inheriting every device from `IDevice`, the benefits of polymorphism allow objects and lists
of objects of this type to be passed around, knowing that the children implement the methods,
but without having to pass them individually. Reflection allows programs to look at and use the
definitions in a class, so objects can be created on the fly, along with method calls and other
functionality provided by the `Class` class. The requirements in `DeviceFactory` are simple
but the functionality is there to execute code without knowing much about it.

5.4.2.3 Database

Along with data generated by the users, the database stores configuration data for the devices.
Java connects to databases using the JDBC API. The noteEd system uses a MySQL database
whose implementation of JDBC is a dependency for noteEd. Rather than having to construct a connection, create a statement and execute a query in every class that requires database access, it made sense to generalise the process by using a class which performed these functions on whatever query it is given.

This enables the database to take advantage of efficiency measures, such as connection pooling, as all connections are handled in one place. The returned result is either a result set for a select query or an integer value for an update query (insert, update, delete) which can then be processed accordingly. Another benefit of this approach is error handling. The classes using the database only have one exception to consider, a custom `DBException`, containing an error message. The class involved with connecting and querying can catch all the exceptions that could occur, and return a readable message rather than exiting abruptly with an SQL error message.

The queries in noteEd are not complex and are written in inline SQL, i.e. embedded in the Java code. This makes maintenance difficult as finding a particular query in a source file is awkward. By placing regularly used queries in a class of their own ensures raw SQL is contained within one class and by using sensibly named methods, means it is easier to find and maintain a particular query.
5.4.2.4 Device Control

In order to call the read method inside a device, at the correct time, each device has an associated instance of a DeviceReader. This class implements the java.langRunnable interface to wait for non-Red5 devices to be active and the java.util.Observer interface so it can be notified by Red5 devices. The methods make several checks to ensure it records the correct stream.

The system is required to allow multiple lectures to be captured simultaneously. The most obvious solution is to provide a thread for each lecture. The DeviceThread class, which extends java.lang.Thread, controls the actions for a single lecture. It contains a list of devices, provided by the factory class, an identifier (the user name of the lecturer), a list of readers, a map of active devices for convenience and a java.util.concurrent.ExecutorService to start and stop readers for this user.

The constructor initialises the fields in the class with default values. The startDevice(...) method sets up a device making it eligible for reading. This includes using parameters from the API and calling the aforementioned prepare method as well as updating the database to store a timestamp which is referred to later when integrating with Synote. The stopDevice(...) method, does the opposite of the start method, as well as updating the database to record the end time of the recording. There are several convenience methods to ascertain whether a device is active and to stop all devices and readers in one go.

As the class is a thread, it is convenient to override the run() method provided by the parent class to instantiate the readers and add them to the executor service which takes care of the concurrency. This method also does one other important job; it adds the DeviceReaders to the class which is observable. This ensures that they receive notifications whenever the observable class notifies its observers.

This brings us on to the DeviceManager. This class is the only way of accessing the device functionality and extends java.util.Observable, allowing it to notify any attached observers when Red5 says there is a stream that has started or stopped. The observer pattern is shown in figure 5.7. The Red5 framework calls the streamBroadcastStart(IBroadcastStream stream) method in the Red5 application when a stream becomes available, this method contains the lines shown in Figure 5.8:

The Observable methods are called which sends the stream to all of the DeviceReaders; they use the stream object to decide if it is the stream for the user they are associated with and act accordingly.

DeviceManager also utilises the singleton pattern (figure 5.9) to ensure only one copy exists in the Java Virtual Machine (JVM) meaning the system does not have to keep track of the particular active manager holding the DeviceThread for a particular user; the thread is guaranteed to be in that particular, singular instance. The implementation of the singleton pattern involves a
Figure 5.7: Observer Pattern [15]

```java
DeviceManager dm = DeviceManager.getInstance();
dm.setChanged();
dm.notifyObservers(stream);
super.streamBroadcastStart(stream);
```

Figure 5.8: Starting a Stream Broadcast

private constructor and an access method to the single instantiation stored in the singleton class. Instantiation can be lazy (when the accessor is first called) or can occur the first time the class is referenced."

Figure 5.9: Singleton Pattern [13]

As well as managing the threads, it provides a server socket which is required for any devices requiring socket communication with the client (notably, screen capture). This socket provides a mechanism to acquire parameters to prepare the device as well as receive data during the reading process. Access to the singleton DeviceManager comes from the final class APILink. It is a number of methods which map to the PHP API functions, ensuring the PHP code can follow the same pattern every time it needs to communicate with Java. It will only call methods in the APILink class; these methods will only throw an APIException if not successful, never another type as all others get caught in the Java code.

As the APILink is the only class ever explicitly referred to, it is sensible to access the configuration file from here, as it can be loaded before any method calls are made ensuring the configuration data is available to anything that needs it (using a public object). The configuration
file contains server settings, file and folder paths and the connection string for the database. It is parsed using Apache’s Commons Configuration, specifically the `PropertiesConfiguration` class. By including a jar file in the project, it allows usage of the classes defined in that jar. Apache have developed a system for processing configuration data which converts the data stored in the file to various Java types. Commons Configuration provides typed access to single, and multi-valued configuration parameters [9].

Some API calls are requests for information and do not require access to other classes. One such call is `getServerConfig` which returns the settings a client should use to connect to the server. These include the port for sockets and the port and relative URL for streams. Most API calls trigger some action in the server; the call is passed on to the manager which handles it.

### 5.4.2.5 Downloading Lecture Multimedia

One such manager is the `FileManager` which is responsible for performing actions on the file system and updating any associated records in the database. This includes creating a folder for each lecture to store multimedia in, retrieving a list of files created and making a lecture package available for download. The download functionality is the most interesting action and requires a number of helper methods and external tools.

Firstly, it generates an XML file of all the notes written in the lecture. Secondly, a summary of the lecture containing important timestamps and the files created. Thirdly, an XML file which can be given to Synote and contains the notes as relative times to the first recording for synchronised playback. This file is generated using an XSLT transform on the previously generated files. All three XML files have a schema to ensure they can be validated and are consistent. There are various ways of using XSLT in Java; the differences depend on the XSLT version required. Saxon can provide schema-aware transforms using XSLT 2.0, whereas Apache’s Xalan is XSLT 1.0 and so is only used if Saxon has a problem.

Once the XML has been generated, each file in the folder for the given lecture is added to a zip archive with the same name. Java has built in classes to create zip files and a `ZipOutputStream` can be treated like any other Java IO stream by writing byte arrays to it. This zip file is moved to the `/downloads` folder inside the web server file tree; lecture material is by default stored outside the web server file structure so it only becomes available when the creator chooses to download.

Finally, the original files are deleted to save space and the database is updated to remove the notes and recordings, with the lecture being marked as downloaded so the housekeeper can delete it and the zip file next time it runs. The `FileManager` is also a singleton so that the folder mappings are not dispersed and the reusable XSLT transformer is not re-instantiated.
5.4.2.6 Error Handling

Due to the distributed nature of the system and the use of threads and concurrency, there are many points of failure and potential difficulties. A server socket blocks whilst waiting for a client to connect; if the client has failed, the thread will be in deadlock, this has to be avoided (using a timeout). There could also be a problem with the database connection.

The error handling on the server took two forms. Firstly, defensive programming on the server so it recovers from any issues it has itself. This is particularly relevant to database and file system accesses. Secondly, protecting itself from failures by a client during recording to avoid loose threads waiting for a termination signal. All exceptions caught are thrown back to the original caller with APIException, the only exception thrown to PHP.

5.4.2.7 Maintaining State

When the stream from Flex is ended, the Red5 framework calls the streamBroadcastClose (IBroadcastStream stream) method in the application. The method checks to see if the device associated with that stream is still active and if so, calls the DeviceManager to get it to stop. This ensures the device, and therefore its reader, is stopped. The Red5 application is also called automatically when the Flex application connects to and disconnects from the server. These are convenient times to store any parameters from the connection and tidy mappings when no longer required.

Red5 also has a scheduling system built in which has been used for the purposes of housekeeping. It would have been possible to develop one using the Java Timer API but the actions to perform are simple so the overhead would be a waste, plus the Red5 implementation is readily available and more powerful. This means the housekeeper is started when the web server is started and its jobs (defined in a configuration file and added by reflection) are called at a specified interval. Each housekeeping module is simple as they just have to keep the database tidy and remove any old or unnecessary files.

The system logs messages when an event happens in the system. This is useful for locating bugs and problems and is a useful trail of events to discover how effective users are at interacting with the system. Logging is at info level when a normal event happens and error level when an exception is caught. There are a number of dependencies and external libraries used in the noteEd server. These include Red5, Xuggle, MySQL, Apache Commons, Apache and Saxon XML libraries, Spring and their associated logging systems including Log4j, which noteEd uses.
5.5 Synote Integration

Synote can only display one video, so the user can choose whether to display the screen or the webcam. However, only the webcam is captured with the audio; the screen is captured as a video generated from images, without sound. It was therefore necessary to perform some post-capture processing on the screen recording to attach the audio track, as shown by figure 5.10. This way the user can choose between videos, as either can be played back with the audio captured from the lecturer’s microphone. The Xuggle module was used on the server to facilitate this media processing. Our class extracts the audio from the microphone stream, and merges it with the video file created from the screen capture.

The streams and notes captured must be exportable to Synote. Once the lecture has finished, the server packages up the notes, video and audio into a compressed Zip file. We translate our full XML output into a format which is already recognised by the Synote system. Synote creates a ‘Synmark’, a synchronised bookmark, for each note created using the noteEd system. Each Synmark is titled ‘noteEd by <username>’, timestamped with the time at which the note was started and the content of the Synmark contains the note as submitted to the noteEd system. The timestamps of each note are relative to the start of the video to ensure they are synchronised correctly by Synote.

![Diagram](image-url)
5.6 User Interface

5.6.1 Development

5.6.1.1 Iteration 1: Prototype

The implementation of the note taking system began with a prototype. It consisted of a simple PHP system that had direct access to the database. This provided a demonstration for the first progress seminar and helped us understand which web interface methods we would need to implement in our API. The API was constructed in parallel as it would contain methods created by all the members of the group. Using the prototype we quickly developed the methods we needed and made use of the API testing tool to check for accuracy. Figure 5.11 is a screenshot showing the prototype design.

![Prototype Screenshots](image)

5.6.1.2 Iteration 2: Styling and API

The second iteration had us create a style for the interface. We began with creating a background theme which had a content pane that changes as the user navigates the site. Afterwards, we developed each page in turn until there was an example for every page on the site. We inserted example values so that we could show the customer what the site would be like. This iteration also meant ensuring the style would be consistent on all web browsers. To do this we used a strict HTML 4 doctype which meant conforming to a set of guidelines that would ensure cross browser consistency.

While the style was developed we also completed the methods in the API. This meant we could use our API testing tool to check that the methods were accurate before integrating them into the interface.
5.6.1.3 Iteration 3: JavaScript Integration

In the final iteration we developed a JavaScript class that would provide functionality to output data from the API to the interface. Here we could make use of our JSON output as it can be quickly navigated in JavaScript. While we developed the web interface we also worked on interfaces for mobile devices. We developed interfaces for both the Android platform and the iPhone. While we developed these systems we enhanced the functionality of the interfaces by introducing quick links to lectures in progress and the ability to create new users. We also integrated the Flex needed to start lectures and begin lecture capture.

5.6.2 Web Interface

The initial page allows the user to log in using their credentials. There are also links to sign up a new account and upgrade an account to lecturer status. All these functions use the same styled inputs for consistency. The upgrade facility requires an extra ‘upgrade code’ which stops unauthorised users from upgrading.

![Image of login and sign up screens](image)

(a) Log on screen  
(b) Create user screen  
(c) Upgrade to lecturer

**Figure 5.12: Log in and sign up screens**
The following screenshots show the navigation screens that allows users to find the right lecture. The entries in the tables can be clicked on to link to new screens. Alternatively, a list of quick links in the left pane can be used to access important lectures. The details of the system and a courses details are shown in the information pane at the top of the box.

**Figure 5.13:** Navigation screens to locate a lecture
The screenshots in figure 5.14 describe the various states of the notes page. The content depends on the state of the lecture and whether the user is the lecturer for that lecture. When a lecture is pending then the only action available is for the lecturer to start a lecture. Once started the lecturer can begin recording and students can make notes. When the lecturer has ended the lecture they can download it. At this point students can look at the notes but not enter any more.

**Figure 5.14: Lecture screens throughout a lecture**
5.6.3 Mobile Interface

The following screenshots show the completed Android and iPhone interfaces.

Figure 5.15 shows the mobile log on screen for both mobile devices. Both screens bear a strong resemblance to the designs in figure 4.6. Both screens show identical components, consisting of the revised noteEd logo, username and password components and a log on button. The mobile log on screen features the same components to the web interface with a slightly different layout.

![Android and iPhone Log Screens](image)

**Figure 5.15: Mobile log on screens**

The mobile course select screen for both mobile devices is shown by figure 5.16. This screen contains the list of courses using all available screen space. Implementation differences between the two devices are visible in the menu bars. We have chosen to follow each device’s default menu appearance as it is important to follow standard practices for each device. However, there is still a strong resemblance between the two devices and the design in figure 4.7.

Figure 5.17 shows the mobile lecture screen for both mobile devices. As with the course list, the content of each device’s screen is very similar. However, another implementation difference between the Android and iPhone application is visible by comparing the title bars. The iPhone allows navigation buttons to be embedded within the screens title bar, whereas the Android application titles can only contain plain text. These screens also follow the designs shown by figure 4.7.

The notes screens shown by figure 5.18 contain the most varied components, and therefore the largest differences between the appearance on each device. However, each screen still contains...
Chapter 5 Implementation

(a) Android

(b) iPhone

FIGURE 5.16: Mobile courses screens

(a) Android

(b) iPhone

FIGURE 5.17: Mobile lectures screens

the same components in the layout specified by the design in 4.8. The back button at the top of the Android screenshot reduces the screen space available for displaying previous notes, however both devices provide a scrollable list view which contains the full note list.
Figure 5.18: Mobile notes screens

(a) Android

(b) iPhone
Chapter 6

Testing

6.1 Overview

It was important to follow a rigorous, validated approach to testing to ensure the produced solution was of industry standard. Our approach to testing was similar to that described by the V-model, shown by Figure 6.1. While the structure of white box tests such as unit and integration tests depend on the internal structure of the system, black box testing was designed to mirror earlier documentation. The system tests were designed to examine each functional requirement elicited during the analysis phase of the project. The user acceptance test was carried out to assess whether the end user believed the goals of our system had been achieved.

FIGURE 6.1: V-model Software Development Process
6.2 Unit Testing

6.2.1 User Interface

To test the Web Interface we had to check that all the functionality worked as expected and that the style was consistent on various browsers. Overall, the tests were successful, but some areas of the project needed minor changes. A few of the more interesting ones are below.

When testing cross platform compatibility, most of the problems were caused by Internet Explorer. To solve this we used a strict HTML 4 doctype which meant a design that worked in one browser would work in all the others. We also encountered a problem interpreting the JavaScript for courses with just one lecture. After investigation we found it was in a missing check in the JSON where we had been treating an element as an array. This was fixed with a simple check for the one element case.

We also found that users could be logged in multiple times, an issue not considered during the creation the API. This was rectified by changing completed API methods and updating their restrictions.

It was also necessary to test the mobile user interfaces created to provide access to the noteEd system. During testing, this module was decoupled from the rest of the system, by storing XML responses to API calls locally. This allowed comparisons between the information displayed on each device for which the interface was implemented. The test scenarios chosen compared; successful and unsuccessful authentication attempts, navigation throughout the course and lecture hierarchy, and note taking attempts on lectures both lectures in progress and finished lectures. Although users could successfully create notes, a number of interface inconsistencies had to be resolved.

If a student navigates to a lecture that is has not been started yet, the interface should prevent the user from even entering a note. However, the lack of lecture status checks could lead to a note being submitted and rejected by the API. Although the system would catch this error successfully, a more intuitive user interface would have prevented the creation of a note entirely. Therefore each interface was adapted to disable the note taking field and submit button on lectures which are not currently in progress.

6.2.2 Media Capture

Whilst developing the Flex streaming interface we used examples which were included with the Red5 server for testing. This verified that the Flex client was successfully streaming the video. Once we had a working streaming application, we also had to perform testing to verify it was calling the API correctly, parsing the XML returned, and reading parameters provided by the web page.
We developed our own server application extending Red5 classes for recording streams from the front end capture. To test this independently, we could initially have used Red5 examples, but the server quickly became specialised to our application. This meant that we had to test it with the actual Flex code, and led to continuous integration testing between the client and server.

Xuggle was required to convert screen capture images into a single movie. Initially, we wanted to include Xuggle on the client side so we could stream a compressed movie, thus saving bandwidth over streaming the individual frames. To test this we ran the simplest possible local socket server to receive the packets. The local socket server then formed the packets into a single file, which it saved locally. This was a success; movies streamed from the applet and were received and saved. These movies were functional and of sufficient quality.

However, if the applet was closed suddenly, or the network connection was lost, the client didn’t send the end-of-file data. This resulted in a file that was unplayable. To work around this, we used the .FLV format, which is designed for streaming, hence we could create files that played. Again the movies streamed were functional. Finally, we had to test this applet in a browser. In this stage the solution failed to satisfy key requirements as Xuggle relies on native code which isn’t portable like Java. We decided at this point that it was not feasible to use Xuggle on the client-side; there were just too many problems.

To fix this, we had to restructure the method of sending screen capture. Instead of streaming a movie file from the client, we stream each captured frame to the server. The server then uses Xuggle to form the movie file. So we tested the applet and Xuggle individually; first testing that our Xuggle class could form a movie file from a stream of images. This was a success and is how we stream the screen capture. With the applet and Xuggle separated, we needed to re-test the applet. Running a modified local socket server, the applet streamed images to the server, which were successfully received and saved locally. Xuggle could then convert them into a movie file of type Windows Media Video or Flash Video.

### 6.2.3 Server Behaviour

#### 6.2.3.1 API

Interactions with the server occurred through the use of a RESTful API. The API (written in PHP) was designed to be highly modular, and loosely coupled. This meant that it was easy to trap and isolate errors. During the development of the API, two strategies were used to ensure errors were minimised. To mitigate the risk of errors, we used combinations of PHP’s assert and var_dump statements to analyse functionality throughout the system. As the methods became finalised, these outputs were only outputted when a field variable (debug) was set to true. Finally, in some cases, testing was able to be automated by utilising the PHP assert method, which ensures variables are equal to a certain value. The combination of these techniques provided
a solution which enabled an efficient method of regression testing as complexity to the system grew.

Within the methods of the API, there was interfacing with MySQL using PHP’s ODBC drivers. In this scenario, it was important to ensure that interaction to the database was secure and did not fall victim to malicious attacks. To protect our system, we incorporated a input validation method, which escapes any quotes from variables. In addition to this, standardised errors are output when database connectivity fails, ensuring a complete system. To test our design, we provided a range of valid and invalid arguments to the API, by varying the inputs, we were able to thoroughly test the overall functionality, thus assuring correct database operation.

While it was important to thoroughly test the API at every level, it was also imperative that we provide development tools for third parties to use when designing interfaces to our system. In addition to an in-depth documentation on the uses and arguments to our API, we also developed an API testing tool, providing an interface to the system where developers could easily customise arguments to the API to find the response they want. This also allowed us to further test our API with a range of inputs under different conditions. By providing the API with both expected and unexpected arguments, and changing the order in which the API was invoked, we could ensure that the API operated and failed with the appropriately output. This was important when creating the user interfaces. Using the API Testing Tool, we were able to see exemplar outputs to different arguments. Aided by this information we were able to detect inconsistencies in our API. One such inconsistency was found when developing the note taking interfaces. We realised that our API produced dates in a different format than they accepted. Similarly, the note visibility field in the database had been misspelt (as visability) causing all subsequent methods to contain the spelling mistake. This was only realised when creating device specific interfaces and shortly corrected. Without the API Testing Tool and the API’s modular design, this task would have been far more difficult. It is a testament to these tools and the design of the system that we were able to locate and correct these errors with such ease.

### 6.2.3.2 Java

Classes were tested in Java by adding a `public static void main(String[] args)` method and either instantiating them or calling their `getInstance()` method if they were singletons. This allowed hard-coded values to be passed to the methods to ensure they behaved correctly for acceptable, borderline and erroneous input. It also allowed checking of return values and catching of exceptions. The addition of output lines was useful for locating the specific line of a bug and conveniently, helped solve problems when combining components later in the project as the value of parameters and return values made it simple to identify logic errors.

The Java code was programmed defensively due to the number of actions reliant on other programs, such as the database or file system. A failure outside the control of the application
sometimes required different commands to be executed in order to complete a request successfully. Irretrievable errors could also occur so the message returned to the API allowed the user interface to provide a suitable message to the user and inform them of the situation.

6.3 Integration Testing

6.3.1 User Interface and API

Integration testing of the user interface required user events to be connected to the noteEd API. Although the majority of API calls returned XML of a similar form to that used in unit testing, there were some exceptions. Since the API queries or updates the database, the database state will be reflected in the XML responses. It was found in some cases that null fields in the database were not added to the XML structure. This could cause exceptions to be thrown on any interface attempting to render these null fields. Checks for null data were therefore necessary for all fields retrieved from the database that potentially could contain no data.

When integrating the Web interface to the API there were several problems accessing the XML. One of these was because of the use of JSON which is used to quickly convert XML into a JavaScript object. We found that the JSON converts arrays of size 1 into a single object which meant that we could no longer use array methods such as length. Also in creating the Web interface we soon found that we needed additional parameters to functions such as getLecturesInProgress. In this case when the user clicked on a course we only wanted to display the lectures in progress for that course. Therefore, we added an extra field to the API which allowed the method to specify a course code.

In the mobile interface development, null checks were also an issue. Despite the fact that Objective-C allows you to send messages to the nil pointer, leaving unchecked nil values would often result in undesired functionality. For this reason, we decided to ignore null data where possible, displaying error messages if there was no possible way to continue with the request. Another issue when developing for the iPhone was the formation of URL’s. It was initially thought that the NSData would automatically encode any URL’s given to it, however, it turned out that URL encoding was required. This was a problem that only arose when making notes with a space. By encoding all the URL’s used in the app, this problem was negated and normal functionality resumed as expected.

6.3.1.1 Applet and Flex Integration

In our design, the applet did not provide any user interface. Instead, all user actions go through the Flex interface. This means communication is required between Flex and Java. Initially, we used the Merapi Bridge, however, this soon became infeasible as there were many dependencies which gave the applet an intolerable file size. This created an issue with slow startups due to
Chapter 6 Testing

the large download. To solve this issue we moved to a new method of communication, using
JavaScript as a reflector to forward messages from Flex to Java. This method was much simpler
to set up, and required no further libraries. However it sacrificed some flexibility which Merapi
promised.

When integrating these sections we discovered another problem with the Java security model.
When we sent messages from JavaScript to Java, the applet no longer sent over sockets. Sending
messages to the Java removed the privileges that signing the applet gained us. To work around
this we had to use an indirect method of calling the start function in the Java, solving the issue.

6.3.2 Java Server and Media Capture Integration

6.3.2.1 Screen Streaming

Integration testing for the screen capture application required the server (an application running
inside Tomcat) to serve a webpage containing our Java applet. The applet was then required to
start streaming images, which the server captured. The server formed a playable movie file of
the images.

A modified applet was created with start and stop buttons and embedded into the page. This
allowed testing without a dependency on the Flex client. The first failure was because of the
Java sandbox. The applet had to be signed so it was ‘trusted’ and can send over the socket.
Once the applet was signed and the user had accepted the security certificate, streaming to the
server was functional.

On the server side, this required a ServerSocket to provide a socket for the client, to allow
the sending of parameters and images. The server code mirrored the client to receive the images
as quickly as they were sent.

6.3.2.2 Webcam and Microphone Streaming

The Flex interface had to receive parameters from the web page, containing information about
the logged in user and lecture to record. These parameters were then used for any calls to the
server API. In order to integrate the Flex front end with the server, a call to the server API started
recordings. This allowed the server to stay in a valid state whilst recording. The server received
API calls to specify which device(s) were recording, as well as authentication details, and which
lecture the user was recording.

On the server side, streams were initially saved in the default directory, to ensure the Red5
application was correctly established. After this was successful, customisation could take place.
### 6.3.3 API and Java

The API communicates with the Java using the PHP/Java Bridge. Setup of this required updates to Tomcat’s lib folder and the addition of provided code to include in the web application. Simple classes at each end were used to ensure an error free channel was in operation. The documentation provided with the library ensured values were converted correctly and exceptions caught when sending data between languages. Once the setup was complete, calls to Java could be added to the API methods safely.

### 6.4 System Testing

System testing comprised of the generation of an individual test-plan to verify each functional requirement had been met. The full system testing table is shown below, along with the result of each test.

<table>
<thead>
<tr>
<th>Req</th>
<th>Name</th>
<th>User</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
</table>
| 1   | Multimedia capture    | Lecturer   | 1. Select lecture  
2. Click start lecture  
3. Select devices to capture  
4. Click start recording  
5. Give lecture  
6. Click stop recording  
7. Click end lecture | Success |
| 2   | Multimedia timestamp  | Lecturer   | 1. Carry out steps 1-7 from test 1  
2. Click download multimedia  
3. Review timestamps in downloaded XML files | Success |
| 3   | Note creation         | Student    | For each note taking interface:  
1. Select ‘in progress’ lecture  
2. Type in a note  
3. Click submit  
4. Review note text and timestamp | Success |
| 4   | Multimedia download   | Lecturer   | 1. Carry out steps 1-7 from test 1  
2. Click download multimedia  
3. Review downloaded multimedia | Success |
| 5   | Synote Integration    | Synote     | 1. Call download lecture API method  
2. Upload downloaded content to Synote | Success |
6.5 User Acceptance Testing

User acceptance testing (UAT) was performed with Dr. Mike Wald and the Synote development team. The test covered lecture multimedia capture, note taking functionality and integration with Synote. The UAT is shown below:

1. Log on
2. Create lecture
3. Start lecture
4. Select to record all devices
5. Start recording
6. Test note taking on web, iPhone and Android interfaces
   a) Log on
   b) Select lecture
   c) Submit note
   d) Log out
7. End recording
8. End lecture
9. Download multimedia
10. Review captured video and XML files
11. Log out
12. Upload multimedia to Synote
13. Playback captured multimedia through Synote

It was planned that the full UAT plan occur in one meeting. However, in reality this was not practical, and the test took place across two separate meetings. The first meeting successfully demonstrated the system’s lecture capture functionality and note taking through the web and Android interfaces. However, the demonstration highlighted an issue related to integration between noteEd and Synote. Although the media was captured correctly, Synote required both the webcam and screen capture to contain the audio track recorded from the lecture’s microphone. Additional coding was needed before the second meeting.

In the second evaluation meeting, the iPhone note taking interface and the combined multimedia capture functionality was demonstrated. Although the demonstration took a number of attempts,
the full capability of the system was successfully displayed. Despite minor instability issues, the acceptance test also highlighted a number of weaknesses in the user interface. While the core functionality was correct in each case, the user interface was not providing intuitive feedback.

Overall, Dr Wald was happy with the functionality provided by our system. He believed it suitably fulfilled the project’s goals and requirements which were negotiated at the start of the project. Mike encouraged us to carry out some scalability testing to check how well the system would cope with the capture of 45 minute lectures and simultaneous lecture capture.

### 6.6 Solution Evaluation

A well structured approach to testing ensured that all the modules of noteEd functioned as a cohesive system. As expected, low level testing uncovered weaknesses within specific units, however inconsistencies between modules were only detected once the modules were integrated together. System and UAT provided our stakeholders with confidence in the system’s functionality, while also highlighting the highest level weaknesses. Overall, we are proud of our system and the end users were very satisfied with our product.
Chapter 7

Project Management

7.1 Project Planning

A project management system was used to track the progress of the project. The system was hosted on a server provided by Liam. The system allowed its users to add tasks, assign them to group members and add a priority and a deadline. This provided us with a system to track which group members are working on what modules, and also to track the overall progress of the project according to original plans.

Gantt charts were created at three stages during the project; at the start of each iteration, providing us with a powerful mix of planning and adaptability. Each Gantt chart focused on only what work was left to be completed.

The Gantt chart created at the start of the project is shown by figure 7.1. It shows a high-level plan for the whole project through from goals to final presentation. The red bars represent the six project deadlines; three seminars and three reports. The green bar towards the end of the project represents the three week winter holiday. The group decided not to schedule work throughout this period for a number of reasons. Primarily, it is not a week of the academic year, and therefore group members are likely to have personal commitments. Secondly, it is also likely that group members will be in different locations, making group meetings and progress tracking near-impossible. It is important to note the reason that individual deadlines have been included on this project Gantt chart. Even though the deadlines apply to individual group members, the task overlaps with preparation for the final presentation and will therefore require time allocation.

A secondary Gantt chart was created at the start of the second iteration as shown by figure 7.2. This chart shows a middle-level plan for the remaining two thirds of the project. The completion of analysis and design phases allows the implementation section to be broken down into individual coding modules. This allows modules of coding to be easily distributed among group members so full advantage can be taken of simultaneous implementation. It is important to note
Chapter 7 Project Management

Figure 7.1: Initial Gantt Chart

that integration testing begins as soon as enough modules are implemented to allow them to function together.

Figure 7.2: Secondary Gantt Chart

At the commencement of the final iteration of our project, a Gantt chart was created as shown by figure 7.3. This chart shows a plan for the final third of the project, focusing on final construction, documentation and a demonstration. At this stage, the bulk of coding has been completed, allowing the additional requirements to be ordered by priority. At this stage the project is still on schedule, avoiding the need to schedule individual work to be completed over the winter holiday. It is important to note that all product development and group report writing has finished by the start of the winter holiday. This allows the group to fully dedicate their time to preparing for the final demonstration.
7.2 Software Lifecycle

It was decided to follow Boehm’s Spiral model to manage development of the project. This was primarily due to the iterative nature of the project schedule. Regular progress seminars were held with project stakeholders to ensure current and future iterations are evolving as planned.

The spiral model also heavily supports the use of prototyping. High fidelity prototypes were used at each stage of the project to demonstrate recently implemented functionality. These prototypes evolved and integrated to form the final product.


7.3 Risk Management

The spiral development model emphasis the importance of risk assessment in iterative software development. To ensure the project follows the initial plan as closely as possible, risk assessment was carried out at the start of each of the three project iterations. References to week numbers can be matched to the individual hours totals displayed by figure 7.9. Full risk assessment tables can be found in Appendix A.

7.3.1 Iteration 1

The first iteration consisted of project planning, analysis and initial prototyping for design purposes. It was concluded by a seminar to check the progress of the project and ensure the solution matches the initial goals. However, a number of risks required suitable management to meet the target of a successful prototype. During week two, Alexander contracted a minor illness causing him to miss a number of group meetings. Despite this, communication between the whole group continued through email, allowing him to work a reduced number of hours from home. Alexander also recognised that he must increase his hours over the following weeks so that each group member contributed equally to the project. This risk was identified as risk 6 of the initial risk assessment. Fortunately, Alexander’s illness was only minor, avoiding the need to adapt the project plan or consider special circumstances.

Another event causing the project to deviate from the original plan was the result of a hardware failure. The group planned to use Matthew’s laptop to deliver the first progress seminar. However, the day before the presentation, a technical fault rendered the screen useless. During the preparation for the progress seminar, the group had allocated a backup laptop which was therefore used to deliver the presentation. This risk was identified as risk 10 in the risk assessment, and was successfully managed through effective contingency planning.

7.3.2 Iteration 2

The second iteration consisted of constructing the final design and a substantial amount of implementation. The iteration was proceeded by the acquisition of a server machine to deploy and test the latest version of our software. Initial testing of the server identified the limit of two concurrent users for the system. This unforeseen constraint did not have a severe effect on the project as described by risk 6 of the secondary risk assessment, as it was mitigated through a higher rate of communication between group members. It was decided that if a group member were to log on to the server outside of a group meeting, the member must be logged on to an instant messaging program. This allowed group members to discuss the priority of their tasks and plan a schedule to complete all planned work.
Another difficulty encountered during the second iteration occurred as a result of a third-party module called Red5. Red5 is an open source Java implementation of a flash media server, necessary to capture the webcam and microphone stream from clients. The Red5 module is almost 60 MB in size, has numerous third-party dependencies and has a steep learning curve. Alexander, supported by two other group members tackled the time consuming task of understanding and integrating the Red5 module during weeks five and six by allocating a larger number of hours to software development. The action taken followed that indicated by the secondary risk assessment.

An additional complication was encountered on the day of the second progress seminar. During database preparation, a system intrusion caused all users to be logged out of the server on which the noteEd system was hosted and a malicious web application was deployed. Although the exploited security hole was closed instantly, the work which was currently in progress was lost. The group worked closely with developers from the Synote project to ensure no other systems were at risk.

### 7.3.3 Iteration 3

The final iteration consisted of implementing any changes highlighted throughout the regression testing process and the construction of a project report and presentation. The most important issue of this iteration was that of prioritising all activity. It was essential that the product was functioning as reliably and efficiently as possible in all required areas. As the project moved towards completion, a large number of features which would improve the usability and desirability of the product were discussed during meetings. This is a common issue of software engineering projects called feature creep. To avoid the project from overrunning or the discussed ideas going to waste, they are briefly described in the future work section of this document.

This project concludes with a final presentation. This presentation will consist of an overview of our experiences of the project and some form of demonstration. At the initial and secondary progress seminars, the group presented a pre-recorded demonstration of the software. However with the system now complete and due to the nature of the product, a live demonstration would convey the power and functionality of the lecture system more effectively. Live demonstrations however are high risk events requiring much planning and preparation. The increased risk involved is largely due to the limitations imposed by the physical infrastructure. The noteEd system is designed to stream multimedia over a network, with the functionality of the system depending completely on the reliability of the Internet connections available. Also, the functionality of the system can be shown by a pre-recorded video, which can look professional and timings will be known, allowing the speech of the presentation to match without hindrance.
7.4 Source and Documentation Control

The ECS Forge Project Management System was used to control source code generated for this project. The repository was also used to control revisions to this document. This allowed the group to track all contributions to every aspect of the project.

The source repository rollback feature was used several times during the project evolution. On one occasion, the initial implementation of a design decision required the refactoring of a large amount of source code. However, this change proved to be too time consuming to warrant full coding and was therefore aborted. The source repository was rolled back to the previous version, ensuring a minimal amount of work was reverted. It was assumed that committed work was in a working state unless explicitly stated in the commit comment, allowing other members of the group to run a different, interacting component safely.

This document was built using the LaTeX document preparation system. This allowed all group members to commit revisions of plain text which would be fully versioned by the repository. Conflicts could therefore be easily identified and the merging of two documents became a relatively simple task. The decision to use LaTeX resulted from the group’s aims to produce the highest quality documentation possible. Although two group members had not used LaTeX before, the other group members were able to pass on previous experience and knowledge.

7.5 Group Work

This project required the even distribution of workload between the five group members. It was important that all group members took part in all aspects of the development process, including design, construction and documentation. However, each group member had their own commitments, both academic and personal, so finding time to meet up required forward planning. To assist with meeting scheduling, a collaborative timetable was constructed detailing every group member’s free and busy time. Despite scheduling a fixed weekly hour slot to meet with our supervisor, it was often necessary to schedule extra meetings, often at short notice. The collaborative timetable greatly simplified this process and allowed us to adapt to recent developments.

An analysis was carried out to investigate how many hours were worked in each week of the project. Figure 7.6 is a line chart representing the group’s hours against the week number. At the time of writing, the project is in its 10th week, so results after this date represent our predictions for the remainder of the project. A general trend can be seen that the number of hours roughly increases as the project develops. A number of peaks are also visible around weeks 4, 8, 10 and 12. These peaks can be attributed to the project deadlines; progress seminar 1, 2, project report and final presentation, respectively.

An interesting metric can be found by analysing when source code was committed to the central repository. Figure 7.7 is a line chart plotting the number of source code commits and hours
worked against the week number. A general increasing trend is visible in both lines, especially after the midway point. A number of other patterns in each line are also visible around the two progress seminars, held at the end of each month. As with the number of hours worked, there are visible peaks on the day of progress seminars. The erratic nature of the committed source code line can be explained by the working week, as there are troughs corresponding to weekends. The largest peak relating to the 05/11/09 is anomalous compared to the rest of the data, but can however be explained by the repository restructuring that took place after the initial prototyping phase.

The time spent by the whole group can be further broken down by considering the interactions within the group and external actors. Figure 7.8 is a pie chart representing how the group’s hours were used. The two most significant sectors, representing individual work and group meetings, each occupy almost half the total time spent each. The two least significant sectors represent
time spent during progress seminars and time spent during supervisor meetings. However, the group meeting total also includes time spent while working in a number of sub-groups in close proximity. We found this to be an efficient use of our time, as it enabled us to effectively distribute the workload while also allowing spontaneous group discussions.

It is also interesting to analyse how many hours individual group members worked each week. Figure 7.9 is a line graph representing hours worked against week number. Clear trends are visible as described in the analysis of figure 7.6, such as the general increase and peaks before deadlines. It is interesting to understand why the hours worked by group members vary
greatly in some weeks and not in others. We believe those weeks, such as week 8, contained a greater number of group meetings. Some lines differ in shape as well as magnitude, such as Alexander's line. His dip during week during week 2 was due to a minor illness, and Alex recognised he would need to increase his hours in the following weeks to ensure each group member contributed equally.

![Line graph to show individual hours worked](image)

**FIGURE 7.9: Individual hours worked per week**

Further individual analysis can be carried out by considering how group members spent their time, as shown in figure 7.10. It is clear that each group member spent an equal amount of time in both supervisor meetings and progress seminars. However, due to Alexander’s previously mentioned minor illness, he missed some group meetings early on in the project. This is reflected by the smaller middle section of his bar. However, the compensation for his illness is visible by the larger top section of the bar. Overall, it is clear that time contributed by each group member was managed efficiently and work was distributed as evenly as possible.

### 7.6 Individual Work

In addition to recognising the group’s achievement, it is important to accredit individuals for specific sections of this project. The following section lists each group member’s individual strengths, sections of project they have contributed to and sections of this document they have written.

When work was distributed between group members, we tried to assign parts evenly. Each member took on roles that they were comfortable with, however each member also often had new technologies or languages to work with. We also divided up the document assigning members those sections that were relevant to the work they had done. However all members contributed to group sections and proof reading.
### Figure 7.10: Breakdown of individual hours worked

![Stacked bar chart to show breakdown of individual hours worked](image)

### Table 7.1: Individual Strength and Contribution

<table>
<thead>
<tr>
<th>Name</th>
<th>Strengths</th>
<th>Coded</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Liam</td>
<td>Web technologies, Flex, RESTful APIs, Java,</td>
<td>Client-side streaming, RESTful API,</td>
<td>System design</td>
</tr>
<tr>
<td></td>
<td>iPhone development</td>
<td>iPhone development</td>
<td>Testing</td>
</tr>
<tr>
<td>Fernando</td>
<td>RESTful APIs, Java, iPhone development</td>
<td></td>
<td>API implementation</td>
</tr>
<tr>
<td>Alexander</td>
<td>Server management, Web services, Java, Flash,</td>
<td>Server-side streaming, Database</td>
<td>Server configuration</td>
</tr>
<tr>
<td>Horn</td>
<td>Database management</td>
<td>management, Synote integration</td>
<td>Streaming</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Implementation</td>
</tr>
<tr>
<td>Oliver</td>
<td>Project management, Mobile development, Java</td>
<td>Web interface, Android development,</td>
<td>Introduction, Analysis</td>
</tr>
<tr>
<td>Parson</td>
<td></td>
<td>Interface design</td>
<td>Literature review</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Project management</td>
</tr>
<tr>
<td>Matthew</td>
<td>Media streaming, Tomcat, Java, Flash, LaTex</td>
<td>Client-side streaming, Web</td>
<td>Review of existing</td>
</tr>
<tr>
<td>Porter</td>
<td></td>
<td>interface, Media transcoding</td>
<td>applications, Analysis of</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>technology</td>
</tr>
<tr>
<td>Oliver</td>
<td>JavaScript, LaTex, Web technologies, Java</td>
<td>RESTful API, Notes capture,</td>
<td>Interface design, Evaluation,</td>
</tr>
<tr>
<td>Wells</td>
<td></td>
<td>Interface design</td>
<td>Interface implementation</td>
</tr>
</tbody>
</table>

---

*Chapter 7 Project Management*  
62
Chapter 8

Evaluation

8.1 Project Outcome

Our task was to complete a system that allows the capture of lecture multimedia over a web interface. By the time of the report hand-in we were able to demonstrate such a system to our customer and show that all the functionality had been completed. We showed that streaming of audio, video and the screen can be achieved and that live note taking was available. Additionally, all multimedia can be downloaded and uploaded into the existing Synote system.

To meet our targets we followed an iterative life cycle model which meant that we produced a series of working prototypes. Each iteration a more complete prototype was demonstrated and any necessary design changes were added to the development in the next cycle. In this way we were able to integrate responses from the customer and work around the problems that we encountered.

8.2 Achievements

Our main achievement was to provide a client server streaming system which operates over a web interface. We integrated multiple languages, employing Flex, JavaScript and Java Applets to enable streaming of multimedia. These front end clients had to communicate with the Java back end which interacted with Red5 and Xuggle and meant overcoming the challenges of controlling sockets and streaming services. Finally our server had to be robust to control saving the data and making it available for download.

In addition to our streaming service we also provided a note taking interface on both the web and mobile devices. This involved developing our API to provide XML responses to queries from these interfaces. Our mobile applications were developed on both Android and iPhone platforms to show the possibilities of our system.
Finally, all the lecture multimedia had to be made available for integration in Synote. We had to develop a process for zipping all the data and combining our XML into suitable forms for the Synote uploader. This section also meant coding a service to add the audio track to the screen capture so that they would play together in the Synote system.

8.3 Critical Evaluation

The strengths of the project was the rapid development and good group collaboration. We worked well as a group and had many group meetings and group coding sessions. When we coded together we avoided getting confused over code versions and could easily query each other if we were stuck. This was important in a large group as it would be easy for one member to get out of sync if we did not meet up regularly.

One of our main weaknesses was the lack of rigorous testing. Due to time constraints we did not get a long time to test the system fully before our user acceptance test. Most of the testing was done by each member on their own code and so meant that it was not a complete test covering all possible scenarios. However, there is still time to do further testing before handing the system over to the customer.

Throughout the project we had good communication with our customer. We could contact them by email and had weekly meetings to discuss our progress. We were given resources when we asked for them and tried to respond quickly to ideas and comments from the customer. We kept a good working relationship between group members and the customer throughout the project and all believe all parties were pleased with the finished solution.

This project has taught us a lot in terms of web technologies and possibilities. We have shown that complex multimedia streaming can be done over a web interface and provided a tool that hopefully will be used to record lectures in the future. Although we could not implement all the functionality a user might want we have included these in the future work section.

8.4 Future Work

8.4.1 Further Integration

The noteEd system was designed to capture multimedia from lectures for playback via Synote. Currently, noteEd allows lecturers to download an archive of all captured multimedia. This archive contains an XML file of lecture notes compatible with the Synote system. The archive also contains video files of the lecturer’s webcam and screen, each with the audio captured from the microphone attached.
Although all this media can be easily uploaded to Synote, this is a manual process which can take a significant amount of time, especially if a large number of lectures need to be uploaded. A useful extension would therefore be to automate this process, so the lecture would be available via Synote as soon as it had finished.

The designed system provides functionality to capture lectures, while Synote provides functionality to playback the multimedia. However, neither system offers the ability to host the audio and video. A possibility would be to extend the functionality of noteEd to host the captured multimedia indefinitely. However, there are obvious scalability issues involved relating to disk space and bandwidth. An alternative would be to make use of an existing video hosting site, such as YouTube or Vimeo.

### 8.4.2 Features

A number of features were suggested during a brainstorm as part of the requirements negotiation process. The richness of notes which could be captured was discussed in detail. Notes written by hand using a graphics tablet would provide students with a high level of freedom. Microsoft Office OneNote is an information gathering package, ideal for taking notes during lectures. Integration with such packages was well outside the scope of this project, however such an extension would greatly increase quality of notes which could be submitted. This feature could be simplified through flattening the handwritten note into an image before submitting it.

A feature which would be of particular use in practice is the ability to mute devices. This would allow the lecturer to send blank audio, video or images to temporarily stop the corresponding device from capturing. This has an advantage over pausing the media stream, as it ensures one file is created for each device of the same duration. Uploading this media to Synote would avoid any complications, as notes uploaded during the muted periods would be attached to the correct period within the media.

The noteEd system currently allows lecturers to capture a video of a single screen; their primary screen. However, many lecturers use a multiple screen set up, generally one screen is used to display the slides and the other to display a presenter view. A more flexible approach would allow lecturers to select which screen, or even both screens from which to record the video.

In the designed system, lecture slides are captured by the screen capture. Although this is a powerful approach as it also supports the capture of live demonstrations, it requires a large bandwidth and a large amount of disk space. An alternative method would be to give the lecture the option of whether to capture the screen, upload the slides or both. However, the more formats of media which can be captured increases the complexity of uploading the media to Synote.

Further features were also considered to provide more feedback to the lecturer about the notes students were taking. This functionality would however depend on the availability of a multiple
screen system. Feedback could take the form of a live stream of notes presented to the lecturer. Another form of feedback could take the form of student polls. A lecturer could pose a question to a group of students and live anonymous feedback could be displayed on the lecturer’s screen. However, such features would require considerable implementation across each mobile interface.

8.5 Final Remarks

In summary, we have completed a successful project, producing a system that meets all of its requirements. We have followed industry standard analysis and design techniques and finished within the timescale and the budget. We have provided Synote a solution for a lecture capture service and recommended several ideas for future work.

We live in an age where people can pause live television, catch up on missed programmes and share information with the click of a button. Although we cannot predict what impact systems such as ours will have in the future we believe that the possibilities are exciting and look forward to seeing what the next ten years will bring.
Bibliography


## Appendix A

### Risk Assessment

#### A.1 Initial Risk Assessment

<table>
<thead>
<tr>
<th>ID</th>
<th>Risk</th>
<th>Probability</th>
<th>Impact</th>
<th>Consequences</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unrealistic project goals</td>
<td>Medium</td>
<td>Severe</td>
<td>Project goals are not met</td>
<td>Project goals are adapted</td>
</tr>
<tr>
<td>2</td>
<td>Unrealistic project plan</td>
<td>Medium</td>
<td>Severe</td>
<td>Project goals are not met</td>
<td>Project goals are adapted</td>
</tr>
<tr>
<td>3</td>
<td>Inadequate requirement</td>
<td>Medium</td>
<td>Medium</td>
<td>Project goals are not met</td>
<td>Project requirements are reassessed</td>
</tr>
<tr>
<td>4</td>
<td>Requirements change</td>
<td>Low</td>
<td>Moderate</td>
<td>System does not meet reqs</td>
<td>Project plan is adapted</td>
</tr>
<tr>
<td>5</td>
<td>Insufficient technology</td>
<td>Medium</td>
<td>Severe</td>
<td>Project goals are not met</td>
<td>Project goals are adapted</td>
</tr>
<tr>
<td>6</td>
<td>Insufficient resources</td>
<td>Medium</td>
<td>Severe</td>
<td>Project goals are not met</td>
<td>Project goals are adapted</td>
</tr>
<tr>
<td>7</td>
<td>Illness</td>
<td>High to low</td>
<td>Low to Severe</td>
<td>Project is delayed</td>
<td>Project goals are adapted to seeking special circumstance</td>
</tr>
<tr>
<td>8</td>
<td>Change in personnel</td>
<td>Low</td>
<td>Severe</td>
<td>Project is delayed</td>
<td>Project goals and schedule are adapted</td>
</tr>
<tr>
<td>9</td>
<td>Project exceeds budget</td>
<td>Low</td>
<td>Moderate</td>
<td>Project is suspended</td>
<td>Resource allocation is reassessed</td>
</tr>
<tr>
<td>10</td>
<td>Feature creep</td>
<td>Medium</td>
<td>Medium</td>
<td>System does not match reqs</td>
<td>Project goals are prioritised</td>
</tr>
<tr>
<td>11</td>
<td>Hardware failure</td>
<td>Low</td>
<td>Low</td>
<td>Work is lost causing delays</td>
<td>Additional hardware required. Adapt plan</td>
</tr>
</tbody>
</table>
## A.2 Secondary Risk Assessment

<table>
<thead>
<tr>
<th>ID</th>
<th>Risk</th>
<th>Probability</th>
<th>Impact</th>
<th>Consequences</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Inadequate requirement</td>
<td>Medium</td>
<td>High</td>
<td>Project goals are not met</td>
<td>Project requirements are reassessed</td>
</tr>
<tr>
<td>2</td>
<td>Requirements change</td>
<td>Low</td>
<td>Moderate</td>
<td>System does not meet reqs</td>
<td>Project plan is adapted</td>
</tr>
<tr>
<td>3</td>
<td>Insufficient technology</td>
<td>Medium</td>
<td>Severe</td>
<td>Project goals are not met</td>
<td>Extra resources used to find work around</td>
</tr>
<tr>
<td>4</td>
<td>Illness</td>
<td>High to low</td>
<td>Low to Severe</td>
<td>Project is delayed</td>
<td>Project goals are adapted to seeking special circumstance</td>
</tr>
<tr>
<td>5</td>
<td>Project exceeds budget</td>
<td>Low</td>
<td>Moderate</td>
<td>Project is suspended</td>
<td>Resource allocation is reassessed</td>
</tr>
<tr>
<td>6</td>
<td>Feature creep</td>
<td>High</td>
<td>Medium</td>
<td>System does not match reqs</td>
<td>Project goals are prioritised</td>
</tr>
<tr>
<td>7</td>
<td>Hardware failure</td>
<td>Medium</td>
<td>Medium</td>
<td>Work is lost causing delays</td>
<td>Additional hardware required. Adapt plan</td>
</tr>
</tbody>
</table>

## A.3 Final Risk Assessment

<table>
<thead>
<tr>
<th>ID</th>
<th>Risk</th>
<th>Probability</th>
<th>Impact</th>
<th>Consequences</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unrealistic project goals</td>
<td>High</td>
<td>Severe</td>
<td>Project goals are not met</td>
<td>Project goals are adapted</td>
</tr>
<tr>
<td>2</td>
<td>Unrealistic project plan</td>
<td>High</td>
<td>Severe</td>
<td>Project goals are not met</td>
<td>Project goals are adapted</td>
</tr>
<tr>
<td>3</td>
<td>Inadequate requirement</td>
<td>High</td>
<td>Severe</td>
<td>Project goals are not met</td>
<td>Project requirements are reassessed</td>
</tr>
<tr>
<td>4</td>
<td>Requirements change</td>
<td>High</td>
<td>Severe</td>
<td>System does not meet reqs</td>
<td>Project plan is adapted</td>
</tr>
<tr>
<td>5</td>
<td>Illness</td>
<td>Low</td>
<td>Low</td>
<td>Project is delayed</td>
<td>Project goals are adapted to seeking special circumstance</td>
</tr>
<tr>
<td>6</td>
<td>Change in personnel</td>
<td>Low</td>
<td>Low</td>
<td>Project is delayed</td>
<td>Project goals and schedule are adapted</td>
</tr>
<tr>
<td>7</td>
<td>Feature creep</td>
<td>High</td>
<td>Severe</td>
<td>System does not match reqs</td>
<td>Project goals are prioritised</td>
</tr>
</tbody>
</table>
Appendix B

Server File Structure

File Store:
C:\noteEdFiles\<courseCode>\_<lectureNo>\_

Web Server:

<table>
<thead>
<tr>
<th>C:\tomcat\</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>lib\</td>
<td>required jar files for all web apps</td>
</tr>
<tr>
<td>webapps\ROOT\</td>
<td></td>
</tr>
<tr>
<td>api\</td>
<td></td>
</tr>
<tr>
<td>methods\</td>
<td>API class files</td>
</tr>
<tr>
<td>php\lib\</td>
<td>PHP utilities</td>
</tr>
<tr>
<td>index.php</td>
<td>API page to connect to</td>
</tr>
<tr>
<td>capture\</td>
<td>Applet and Flash app</td>
</tr>
<tr>
<td>downloads\</td>
<td>end location for lecture zip packages</td>
</tr>
<tr>
<td>images\</td>
<td>images</td>
</tr>
<tr>
<td>java\</td>
<td>PHP/Java Bridge classes</td>
</tr>
<tr>
<td>pages\</td>
<td>PHP pages</td>
</tr>
<tr>
<td>schemas\</td>
<td>XML schemas</td>
</tr>
<tr>
<td>scripts\</td>
<td>JavaScripts</td>
</tr>
<tr>
<td>styles\</td>
<td>CSS and XSLT stylesheets</td>
</tr>
<tr>
<td>WEB-INF\</td>
<td></td>
</tr>
<tr>
<td>cgi\</td>
<td>PHP/Java Bridge launcher files</td>
</tr>
<tr>
<td>classes\</td>
<td>Red5 class files and application definitions</td>
</tr>
<tr>
<td>lib\</td>
<td>required jar files</td>
</tr>
<tr>
<td>index.php</td>
<td>home page</td>
</tr>
<tr>
<td>favicon.ico</td>
<td>icon for browser</td>
</tr>
</tbody>
</table>

**Figure B.1:** Server File System