CA326 Functional Specification

AnswerBox, by Cathal Conroy and Christopher Durning

Table of Contents

1. Introduction

1.1 Overview

Like many others, we believe that when it comes to studying for an exam, practice is absolutely essential. For the most part, there is no better way to reinforce a concept in your memory than by taking the time to try and solve actual exam questions. It is even better if these questions are of the same format as those in the exam. Unfortunately, it is not always possible to source a correct answer to a specific question, and it can often be dangerous to trust answers found online (where there is little to no moderation), or those provided by friends and classmates.

Of course for secondary school students, the obvious solution would be to make the most of your teacher, and have him/her correct your answers. Unfortunately however, the modern classroom is worryingly overcrowded, where a single teacher supervising thirty students is perfectly normal. With thirty students constantly handing up answers, each of which take huge amounts of time to thoroughly check, you could very well be left waiting for over a week for a response, by which time you have probably changed topic altogether.

We intend to build a mobile application on the Android platform which will allow second level students to collaborate by solving past paper questions, and submitting them for evaluation. Think of Stack Overflow. It is essentially a community driven, self-moderated Q&A forum, and has grown to be arguably the world's most popular programming reference site.

The creators of Stack Overflow make little to no contribution to the site's answers themselves. Instead, they have developed their site in such a way as to encourage their users to contribute whenever they can, through the use of visual awards (badges) and increased moderating power over the site. These rewards will be earned by reaching certain amounts of reputation (a simple number associated with your account, increasing with site contributions). We can provide similar incentives within our app to encourage user activity.

Users of this app will be allowed to do two things;

- Browse through answers submitted by other students, voting them up/down, and replying to them. Users will gain reputation for replies.
- Submit own answers to questions. Users will earn reputation as their solution gains up-votes.

All of the application's content will be hosted on a remote Linux server, and will be retrieved dynamically as needed. Users will have the option to download answers they like for offline use.

Users will be forced to create an account in order to access our application. We believe this is the best route to take as it will encourage users to participate with their peers around the country. Of course it could be seen as an inconvenience to some, but much less so than paying up to thirty euro for a hard-back book. We have made the registration process as painless and un-intrusive as possible.

1.2 Business Context

The sample answer rating system we are creating has the potential to draw plenty of attention from many educational companies in Ireland such as Edco, EducatePlus, and especially Revise Wise - a company which dedicates itself to writing fully worked solutions to past paper exam questions. The goals of Revise Wise and our app are very similar. We are both providing sample answers to official leaving cert and junior cert state examination commission papers, and we both aim to provide students with a reliable resource they can study.

Our target market would be Irish secondary school, a population of approximately 345000 pupils, plus the 735 teachers who might find it useful for showing their students an example of what a good answer should look like.

In terms of producing capital, we could offer non-intrusive advertising space throughout our application. Seeing as our app would be accessed by such a specific audience, it would be great for businesses who sell anything from pencils to extra-curricular work books.

1.3 Glossary

API, Application Programming Interface: A layer of abstraction designed to provide an easy-to-use set of tools which often hide more complicated, low-level code.

JSON, JavaScript Object Notation: A lightweight and human-readable data-interchange format. We use this to transfer information from PHP to Java.
PHP, Hypertext PreProcessor: An open-source, general purpose scripting language often used in web applications.
Lazy-loading: A design pattern which dictates that content should be loaded only when it is absolutely needed.
LAMP stack: LAMP stands for Linux, Apache, MySql and PHP. This is a bundle of open-source software collectively referred to as a stac, and used as a platform to host web applications.
HTTP, HyperText Transfer Protocol: A communication protocol for distributed systems.
SSL, Secure Sockets Layer: A standard security technology used to secure communications between a web server and clients.
bcrypt: A password hashing function based on the Blowfish cipher.
VPS, Virtual Private Server: As the name suggests, a virtualized server. In practicality, these function identically to full standalone servers, except less powerful.
MySQL: An open-source relational database management system. We use this to store application content.

2. General Description

2.1 Product / System Functions

Creating accounts

Users will need to create an account, and their information will be stored on our server. Passwords will be hashed using bcrypt.

Authentication

In order to use the app, users will need to authenticate themselves with an email and a password. Our app will then verify the user's credentials by querying the web API.

Updating account information

Users will be able to modify certain account information from the profile tab once they have signed into the application. This will be performed by querying the web API.

Downloading content

When using the application, users will need an active internet connection as all user content will be stored on a remote server. We will implement a *lazy-loading* design pattern for images as not all mobile devices are fast enough yet to justify downloading files the user may not need.

Searching for solutions

Upon entering a set of search parameters, our API will return all textual results via JSON, and image files directly over HTTP in default format.

Storing content for offline use

Users will have the option to save their favourite answers for offline use. We will make use of the Android file system to accommodate this.

Web API

All user information will be stored in a MySQL database running on the same remote server. As it is dangerous security-wise to connect directly to a database, we will implement an API in PHP to deal with any queries the app may need to make.

Remote server

Our remote server will be running a LAMP stack, as we only need a very minimal amount of software to host all of our content. It will communicate to our application over HTTP. We may secure this with SSL if we feel it is necessary.

2.2 User Characteristics and Objectives

The main groups of user for our app would be Irish secondary level students and Irish secondary level teachers.

Students will make up the bulk of users on the app, so their needs will definitely be high on the priority list when designing this app. With students being brought up using smartphones and computers, their knowledge of how apps and software work is very adept. We will need to make sure our app runs as efficiently and as smoothly as any other app available on the market. As technology is so fast nowadays, students do not like it when it takes them multiple clicks to achieve their desired objective. With this in mind we will need to ensure the app design allows the user to get to any answer or page on the app with very few clicks. An option to get a short summary of an answer before opening it is also something that will be implemented. This will save users time so they won't have to open every single answer to see what content it has inside.

The app will also be required to have a clean and simple design as students are easily put off by things that do not appeal to them visually.

When it comes to doing pretty much anything young people can be very lazy and unmotivated, unless they can see themselves earning a reward.

The reputation features which includes getting achievements and up votes from other users is something that will make students want to use the app. for example, Yik Yak and Stack Overflow have communities that are completely driven on a reward system.

Teachers on the other hand would have varied knowledge of technology depending on age and interest. Younger teachers that have a better knowledge of how smartphones and apps work would be more inclined to use our app than older teachers, and the features that appeal to students would also appeal to them. Teachers with less of a technical ability may

also want to use the app, so it is important that we make the app as simple to maneuver through as possible, with not too much excess information on each page.

The search bar feature will help teachers with limitations, as they can just type in what they are looking for in plain text at the top of any page and click search rather than moving through the app the traditional way.

Everything submitted on the app will be monitored using report buttons. Teachers will only want to use the app if all the content on there is strictly based on answering a question, so this feature is very important.

2.3 Operational Scenarios

User registers

When the app is first opened the user is given the option of logging in or registering. The user must create an account if they do not already have one. After the user clicks on the register button a page asking for user account details needs to be filled out. The user will first be asked to enter a username. The username they enter must begin with a letter and have between 3 – 20 characters. If the username they have entered does not meet these requirements a message will appear telling them that they have entered the wrong details and to try again. The user must have an email in order to sign up. Once the user has entered their email they will be required to enter a password. The password must be a minimum of 6 characters and contain at least a letter, number and symbol. If the password they have entered does not meet these complexity requirements the user will be asked to enter another password. To make sure the user has entered the password they meant to, they will have to re-enter that same password in a different text field. When all text fields are filled out correctly, the user can click on create account and they will be brought to the login page. An email containing a unique link will be sent to the user, which they must open to complete their registration.

USE CASE 1	Register	
Goal in Context	User wants to create an account	
Scope & Level	System, Core.	
Preconditions	User must have an email	
Success End Condition	An email containing a unique registration link has been sent to the user's email	
Failed End Condition	The user's email is already linked to an account, or the password's complexity is not great enough.	
Primary, **Secondary Actors**	User Server	
Trigger	User clicks on "register" button	
DESCRIPTION	**Step**	**Action**
	1	User enters their username into the username text field
	2	User enters their email into the email text field
	3	The user enters a password of their choosing into the Password text field
	4	The user enters the same password again into the Re-enter password text field
	5	User clicks "create account" button
	6	If the password the user entered meets the complexity requirements, the new email and password are stored in the database
	7	User has successfully created an account and will now be brought to the login page
EXTENSIONS	**Step**	**Branching Action**
	4a	If the password the user has entered does not meet the complexity requirements, tell user to enter a password that is adequate.

User logs in

The user can only log in if they have an account already made. On the first page of the app the user clicks the login button, then a new page appears asking the user to enter their

existing account details. The user must enter their email and password in the text fields provided. If the email or password they have entered does not match what is in the database then they will be informed that the details they have entered are incorrect. A "forget password" option will also be available. When this button is clicked a link will be sent to the user's email which will bring them to a webpage where they can set up a new password. Once the details they have entered are correct the user can click sign in and they will be brought to the home page.

USE CASE 2	Login	
Goal in Context	User wants to login.	
Scope & Level	System, Core.	
Preconditions	User must have an account already made.	
Success End Condition	User's email and password are accepted and they are logged in.	
Failed End Condition	User's email or password are rejected and they are not logged in.	
Primary, **Secondary Actors**	User Server	
Trigger	User opens application.	
DESCRIPTION	**Step**	**Action**
	1	User clicks on "log in" button.
	2	User enters their email in the email text field
	3	User enters their password in the password text field
	4	User clicks on "sign in" button
	5	If a matching set of credentials are found in our database, the user is logged in.
EXTENSIONS	**Step**	**Branching Action**
	5a	If the email or password the user entered does not exist in our database, tell user that what they entered is incorrect
VARIATIONS		**Branching Action**
	1a	User clicks on "register" button to create an account (Use Case 2)
	2a	User clicks on "forgot email or password" (Use Case 3)

User forgets password

If the user cannot remember their password, they will need to request a password change. At the login screen, they will have the option to do so. By clicking "forget password", they will be prompted to enter their email address. If the email address exists in our system, they will be sent an email containing a unique link with which they can follow to reset their password. The link will lead to a basic form on our server with a password field. If the email they entered does not exist, they will be informed and no email will be sent.

USE CASE 3	Forget password	
Goal in Context	User wants to change their password	
Scope & Level	System, Core.	
Preconditions	User must have an account already made	
Success End Condition	User successfully creates a new password	
Failed End Condition	User has not successfully created a new password	
Primary, **Secondary Actors**	User Server	
Trigger	User clicks on “forget password” button	
DESCRIPTION	**Step**	**Action**
	1	User enters their email address into the text field provided
	2	User clicks on a button which send link to the user’s email address
	3	The user clicks on this link which opens a webpage asking them to enter a new password into a text field and to re-enter it.
	4	User enters new password into the text field and clicks “submit”
	5	The password for this user is updated in our database
	6	User has successfully changed their password
EXTENSIONS	**Step**	**Branching Action**
	4a	If the password the user has entered does not meet the complexity requirements, tell user to enter a password that is adequate.

User browses answers

On the home page the user decides what answers they are looking for by selecting a year and a subject from drop down boxes. They then proceed to the results page by clicking a button that determines the level of the answers they are looking for, either higher or ordinary.

USE CASE 4	Browse answers	
Goal in Context	User wants to find an answer	
Scope & Level	System, Core.	
Preconditions	User must be signed in	
Success End Condition	The answers the user searched for are displayed	
Failed End Condition	The answers the user searched for are not displayed	
Primary, **Secondary Actors**	User Server	
Trigger	User clicks on "H" or "L" button	
DESCRIPTION	**Step**	**Action**
	1	The user chooses a "subject" from the first drop down list
	2	The user chooses a "year" from the second drop down list
	3	The user clicks on either the "H" or "L" button, which represent the level of answer the user wants returned to them
	4	User is brought to the results page where they can look through answers.(Use Case 6)
VARIATION	**Step**	**Branching Action**
	1a	User searches for answer using search bar (Use Case 5)

User searches

The user can search for an answer or another user's profile very quickly by using the search bar. The search bar can be accessed by clicking the search icon at the top right of every screen.

USE CASE 5	Search	
Goal in Context	User wants to find answers	
Scope & Level	System, Core.	
Preconditions	User must be signed in	
Success End Condition	The answers the user searched for are displayed	
Failed End Condition	The answers the user searched for are not displayed	
Primary, **Secondary Actors**	User Server	
Trigger	User clicks return after entering text in the search bar	
DESCRIPTION	**Step**	**Action**
	1	The user clicks on the search bar at the top of the page
	2	The user enters the title of an answer and presses return
	3	A list of suggested answers are taken from our database and displayed on the screen.
	4	When the user clicks on one they will be brought to that specific answer page
EXTENSIONS	**Step**	**Branching Action**
	2a	If the answer they searched for doesn't exist the search will return empty and they will have to enter a different search

User submits an answer

On the results page the user also has the option to submit their own solution. They can do this by clicking the submit button at the top of the page. The submit button brings the user to another page which consists of a title text field and a body text field. The user can add text to these boxes and also add images from their camera or photo library by clicking the add images button. A preview button is found at the bottom of the page to allow users to preview their answer before submission and then the submit button is used to submit the answer.

USE CASE 7	Submit answers	
Goal in Context	User wants to submit their own answer	
Scope & Level	System, Core.	
Preconditions	User must be signed in	
Success End Condition	The answer is has been accepted	
Failed End Condition	The answer has not been accepted	
Primary, **Secondary Actors**	User Server	
Trigger	User clicks “submit” button on results page	
DESCRIPTION	**Step**	**Action**
	1	The user adds the title of the answer in the first text field
	2	The user adds the content of the answer to the second text field called body.
	3	The user can also add images to the body by clicking on the button “add images”
	4	The user can preview the answer by clicking the “preview” button, which will show the user how the answer will look on the results page
	5	The user can successfully submit the answer by pressing the “submit” button located beside the preview button
EXTENSION	**Step**	**Branching Action**
	1a	The user can exit this page and go back to the results page by pressing the “x” button in the top left

User saves an answer

All saved answers can be found on the saved page. The user accesses this page by clicking the an icon on the menu at the bottom of the page. Once the user clicks on a saved answer a page opens containing the title, body and images for that answer. An exit button can be found on the top left of the page which will allow the user to go back and browse through other saved answers.

USE CASE 8	Save answers	
Goal in Context	User wants to save answer to their saved page	
Scope & Level	System, Core.	
Preconditions	User must be signed in	
Success End Condition	Answer has successfully saved to the saved page	
Failed End Condition	Answer has not successfully saved	
Primary, **Secondary Actors**	User Server	
Trigger	User clicks “save/download” button below an opened answer	
DESCRIPTION	**Step**	**Action**
	1	User can scroll through answers on the saved page
	2	When the user clicks on an answer, a page opens containing the answer content, and there is no buttons or comment section is found below
EXTENSION	**Step**	**Branching Action**
	2a	The user can press a back button on the top left of the page which will allow the user to go back and browse through other saved answers.

User profile page

Users can access their profile page by pressing "me" on the menu located at the bottom of the page. The page contains your reputation, account information settings and a logout button.
When the user clicks on the profile button a page opens contain all the user's achievements earned from submitting answers and getting up votes. The user can get out of this page by pressing the back button. Once back on the "me" page, the user can access settings to change account information such as their password. The logout button on the "me" page allows the user to sign out. When the login button is pressed the user is brought back to the log in or create account page.

USE CASE 9	Profile page	
Goal in Context	User wants to access their profile page	
Scope & Level	System, Core.	
Preconditions	User must be signed in	
Success End Condition	User has logged out of their account	
Failed End Condition	User has not successfully logged out of their account	
Primary, **Secondary Actors**	User Server	
Trigger	User presses "me" button on menu located at the bottom of the page	
DESCRIPTION	**Step**	**Action**
	1	The user can click on "change password" will allow the user to change their password
	2	The user will be required to enter their old password before they can create a new password
VARIATION	**Step**	**Branching Action**
	1a	User can look at their profile by clicking on "profile" which will display the users achievements
	1b	The user can logout by clicking the "logout" button

2.4 Constraints

Server resources

The VPS we are using as our server is limited to 512MB of physical memory and 20GB of persistent storage. While this is more than enough for the time being, it may not be enough if the user base grows large enough. The cloud platform our server is hosted on allows for very easy expansion if needs be, but for the purposes of this project this shouldn't be necessary.

Data storage

In the real world, we would need to think about server and data replication to bring our system to as close as zero-downtime as possible. We believe this is out of the scope for our project and so we will not be implementing any backup functionality.

Time

The time allocated by DCU will be enough to develop the application and server, however it will not be enough to grow a user base. While this will not affect our technical demonstration in any way, it will mean that the application won't *feel* complete, as user contributions are a fundamental aspect of our idea.

Content

Another result of having no user base is that there will be little to no content in the application. We will have two options; to either fill the app with dummy content to demonstrate the different functionalities, or to possibly speak with a teacher in school and ask if they could provide sample, high quality answers. The latter will likely be the route we take, as both of us have brothers/sisters in sixth year studying for the leaving cert.

Moderation

As our application will allow users to upload image files, we will need to find a way of moderating content. In a best case scenario where we have thousands of users with potentially hundreds of images being uploaded on a regular basis, moderating all content ourselves would be impossible. Instead, we will allow the application to be self-moderated. We will provide users with a way of reporting content which they deem inappropriate, and we intend to give users who earn high levels of reputation certain privileges, such as being able to respond to these reports.

Testing

This may well prove to be one of the most challenging parts of our development process. Although we did have a software testing module in second year, it was based around a very simple terminal program. It was very easy to test it thoroughly. Android however is a massive framework where any number of things can go wrong with even the smallest amounts of custom code. We will need to perform a great deal of research to figure out how to implement effective tests for our app. We can of course still perform manual testing. This involves the user actually navigating through the application as if they were a person really interested in using the app. We could do this ourselves, and with the help of our friends or family; people who don't understand the ins and outs of the application.

3. Functional Requirements

3.1 User login

- **Description:** Our application will not allow anonymous users. You will not be able to access the app's contents without a valid account. Users will have to authenticate themselves with an email and password.
- **Criticality:** Our app relies on user contributions. We cannot reliably identify unique users without a login system. It is therefore a feature we consider highly critical.
- **Technical issues:** We may decide to use SSL for client/server communications, however this is not essential.
- **Dependencies:** The user must have created an account in order to login. This is also the only entry point to the application, and so all other areas of the system except registration rely on a successful login.

3.2 User register

- **Description:** When registering a user passes a made up username, their own email address, a password and a re-entry of the password in order to register for the app. These details are linked together in the database so that when decides to login the system can give them access to the account they made.
- **Criticality:** This is another highly critical component of the system for the same reasons as the login system.
- **Technical Issues:** It is also important that passwords stored in the database are encrypted using bcrypt, a password hashing function that scrambles each password and adds random characters, which will make it very difficult to find out what the original password was even if the database is somehow breached. Once a new username is entered in the text field, a search for this username in the database must be carried out and if it already exists, the username entered will have to be changed.
- **Dependencies:** Without users being able to register there would be no system in place to allow users to login using account details. The registration page provides the database with the account details of every user. The app needs this requirement in order to be accessed.

3.3 Browse answers

- **Description:** Users will need be able to find the answers they are looking for with as few clicks as possible. We decided the quickest way to narrow down search results would be to first define your subject from a dropdown, then enter the year, and finally click H for higher level or O for ordinary. Answers will be pulled from the database and displayed in order of votes. That is, the highest rated answers for a specific subject/year will appear at the top of any result list. Each item in the result list will display certain key details about the answer - the title, the uploader and the rating. Users will have to click on this item to open the answer fully.
- **Criticality:** Once again, browsing answers is a truly core concept in our app - without being able to browse through user submissions, there is really nothing left.
- **Technical issues:** None.
- **Dependencies:** The user must be logged in.

3.4 Submit answers

- **Description:** A user can submit an answer once they press the submit button on a results page. A submit page consists of a title text field, a body text field and a button that allows users to upload images.
- **Criticality:** A critical feature. Users need to be able to submit answers if they wish. Without this feature, there is no application.
- **Technical issues:** None.
- **Dependencies:** The user must be logged in.

3.5 Rating answers

- **Description:** A key feature in our app is that users have the ability to rate any answers they find up or down. In doing this, the *quality* of the answers can be judged somewhat without any direct input from the developers. We are hoping that the collective judgement of many users compensates for the fact that the answers is not being scrutinized by a teacher.
- **Criticality:** This is a key feature, because without it there is no way of knowing how to order answers in search results.
- **Technical issues:** None.
- **Dependencies:** The user must be logged in.

3.6 Answer search

- **Description:** All the answers submitted by users are stored in a database, separated by different tables, entities and attributes. This requirement will allow a search from most pages on the app and will browses through all the answer information in the database and get immediate and specific results from it based on the string used in the search bar.
- **Criticality:** This feature is not essential. We have designed our app in such a way that the user should be able to find the answer they are looking for in as few clicks as possible. This feature will be one of the last we develop and only if we have time.
- **Technical issues:** When creating this requirement, we will need to make sure our SQL code securely holds strings securely and doesn't allow users to gain any access to our database by having type-safe SQL parameters. Suggested searches may also be implemented.
- **Dependencies:** The user must be logged in and on a page where the search bar is available.

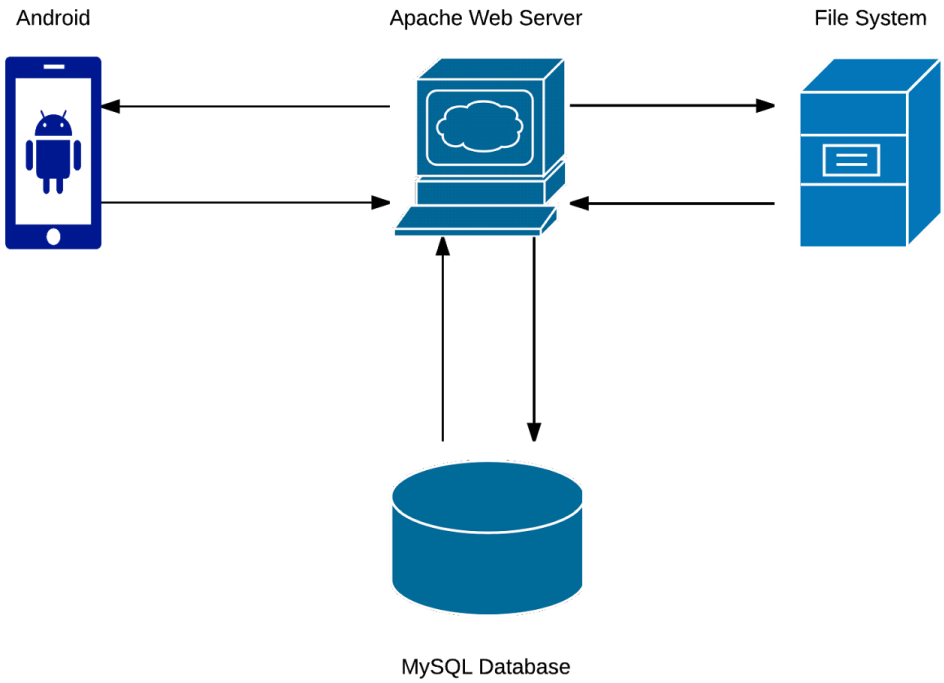
3.7 Saving answers

- **Description:** Answers that are saved by a user are saved only to that user's account. This allows users to access them specific saved answers quickly and whenever by pressing the save icon which will bring them straight to the saved page.
- **Criticality:** It's crucial that users can save answers so they can keep a library of answers that applies to their subject. This will stop them having to search through many different submissions in order to find what they are looking for.
- **Technical issues:** None.
- **Dependencies:** The user must be logged in to access the saved page and they must have previously searched and found answers they liked and saved them.

3.8 Edit settings

- **Description:** When users navigate over to the profile tab, they will have the option to change several settings about their profiles. For example, their password or whether they're studying for the junior cert or leaving cert.
- **Criticality:** This is not critical to the running of the app at all. Of course it's nice to be able to change your password, but the application could perform everything it needs to without these options.
- **Technical issues:** None.
- **Dependencies:** The user must be logged in.

4. System Architecture



The figure above describes how our various systems will communicate at a very high level. When the Android application needs to query the API for textual or binary data, it must make a HTTP request to the Apache web server which hosts the API. The file system and the MySQL database are installed on the same machine as the web server. We do this for two reasons.

Firstly, as there is no networking between the systems, there is essentially zero latency. Modern mobile users have grown to be impatient with modern technology, and so we need to accommodate this as much as possible. Unfortunately there will of course still be lag between the mobile client and the web server, but there is simply nothing we can do about that.

Secondly, data storage on mobile devices is often very limited. It is unfair to fill a user's device with data that they do not absolutely need. By requiring an active internet connection (with the exception of saved answers), the user can ask the server for exactly what data it needs, when it needs it.

Android

This is the only part of the system architecture which the user interacts directly with. All content will be displayed and modified through the application. Any persistent changes made to the data will be communicated to the server through an API.

Apache web server

This will be installed in a Linux environment, specifically Ubuntu 14.04 LTS. We chose to use Apache as it is the only free and open-source web server either of us have any experience with. Although it is easy to configure a basic server, we are helped by the fact that our Networking module covers Apache installation and post-installation procedures, such as setting up a secure firewall.

We will build an API in PHP to run on this server, to interpret requests from Android clients and serve them accordingly. We will need to learn how to safely read and write to a MySQL database with PHP, preventing common security holes such as SQL injection.

MySQL database

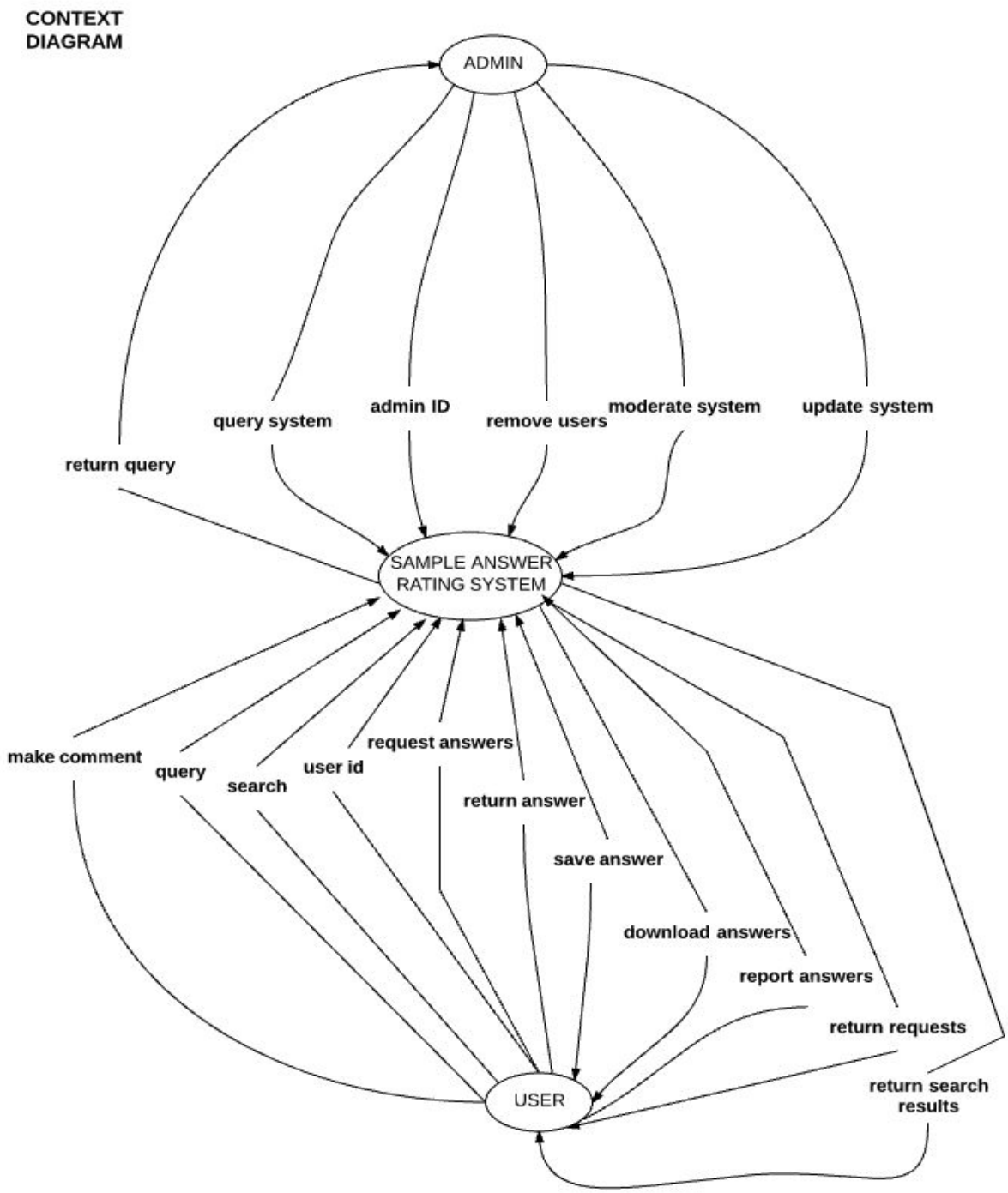
We will of course need a database management system to organize our data for easy and efficient manipulation. We chose to use MySQL as it is free, open-source, and it is what we used ourselves in our second year database module.

File system

We will store all images and other uploaded files directly onto the storage used by the operating system. All files will be given unique names, identifiable only by records in the database.

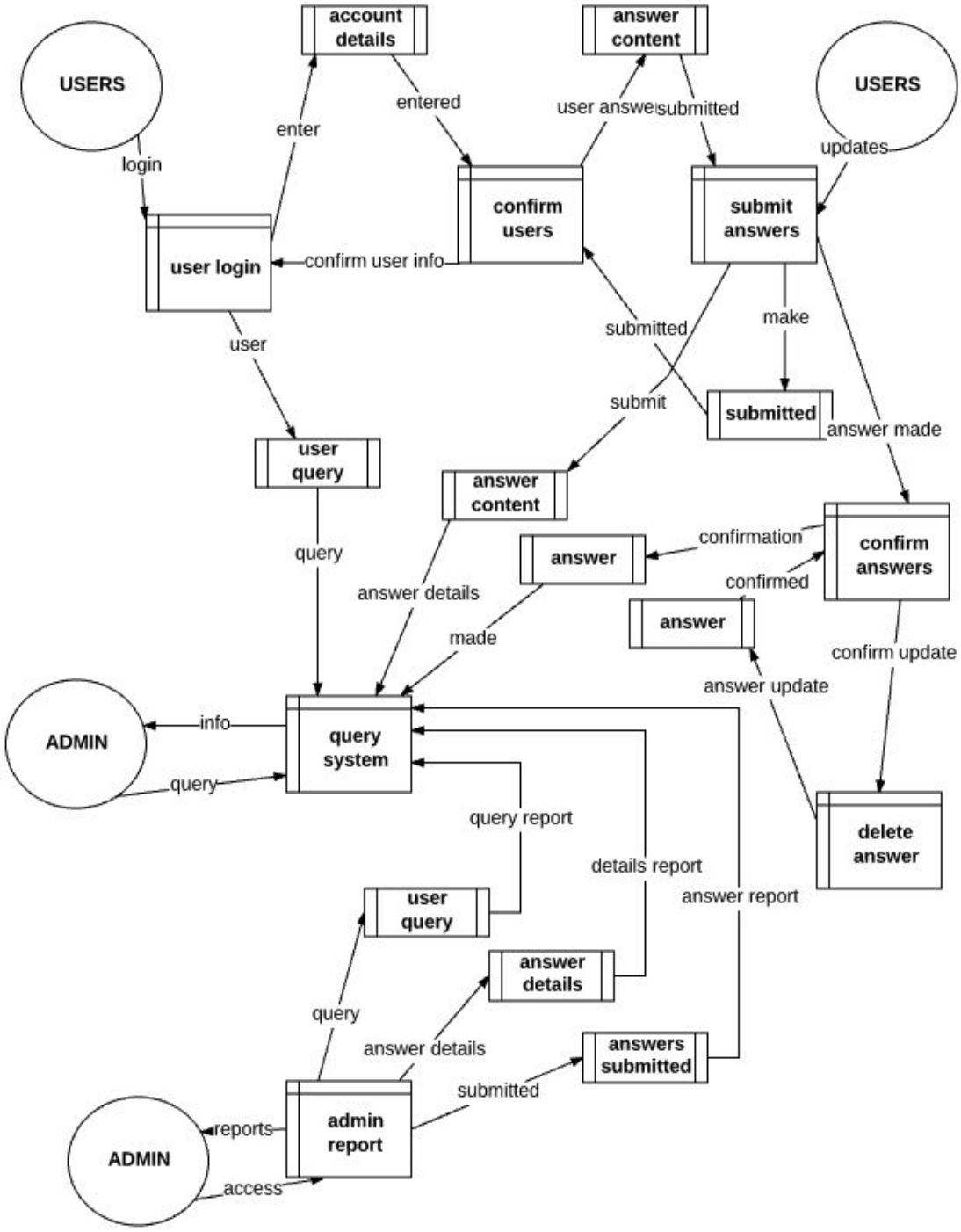
For example, if a user opens an answer which contains an image, the client will make a HTTP request to the web server. The request will be answered by the web server and dealt with by the logic we have implemented in PHP. The API will contact the database to find out where in storage the image is held. Once it finds the image, the web server will reply to the client with the image file and whatever other textual information was needed to display the answer.

5. High-Level Design



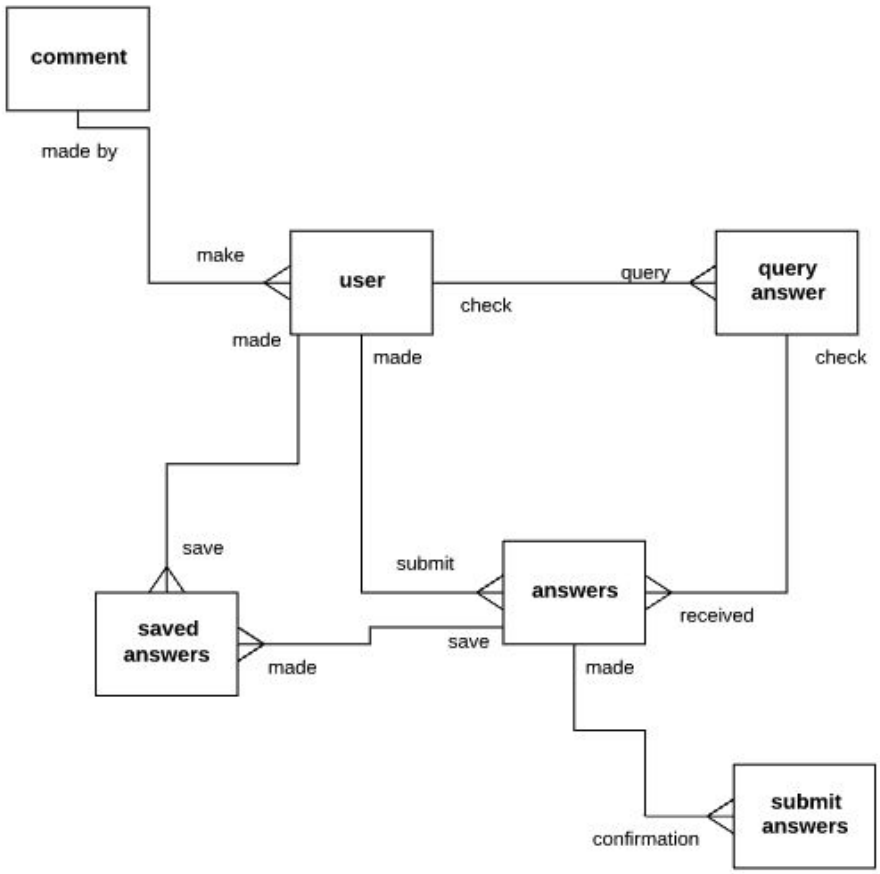
This diagram shows how the system interacts with its external entities.

DATA FLOW DIAGRAM



This diagram shows the flow of data through the system.

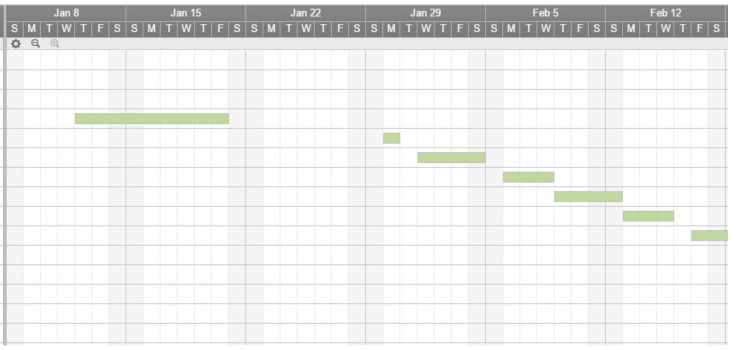
LOGICAL DATA MODEL



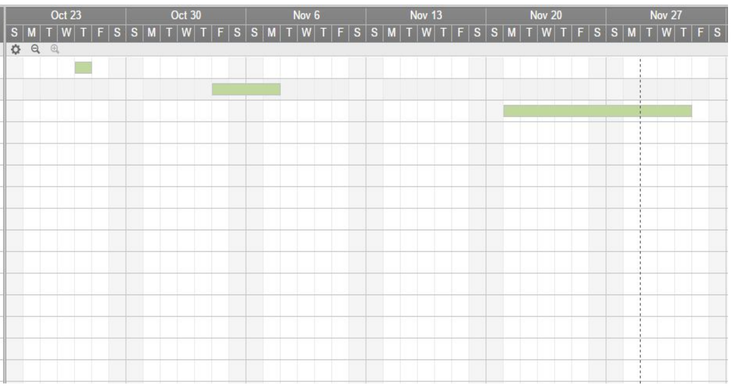
This diagram shows how users and the system interact with each other through cardinalities.

6. Preliminary Schedule

Task Name	Duration	Start	Finish	Status
Proposal	1d	10/27/16	10/27/16	Complete
Presentation	2d	11/04/16	11/07/16	Complete
Functional Specs	9d	11/21/16	12/01/16	In Progress
Exams	7d	01/12/17	01/20/17	Not Started
Design Database	1d	01/30/17	01/30/17	Not Started
Login / Register	4d	02/01/17	02/04/17	Not Started
Android HTTP Client / API	3d	02/06/17	02/08/17	Not Started
Home screen & bottom nav bar	3d	02/09/17	02/12/17	Not Started
Server API	2d	02/13/17	02/15/17	Not Started
Browse answers	4d	02/17/17	02/21/17	Not Started
View answers	1w	02/21/17	02/28/17	Not Started
Save answers	3d	02/23/17	02/26/17	Not Started
Update profile	3d	02/27/17	03/02/17	Not Started
Final testing, anything else not fin		03/03/17	03/09/17	
Project Deadline	1d	03/10/17	03/10/17	



Task Name	Duration	Start	Finish	Status
Proposal	1d	10/27/16	10/27/16	Complete
Presentation	2d	11/04/16	11/07/16	Complete
Functional Specs	9d	11/21/16	12/01/16	In Progress
Exams	7d	01/12/17	01/20/17	Not Started
Design Database	1d	01/30/17	01/30/17	Not Started
Login / Register	4d	02/01/17	02/04/17	Not Started
Android HTTP Client / API	3d	02/06/17	02/08/17	Not Started
Home screen & bottom nav bar	3d	02/09/17	02/12/17	Not Started
Server API	2d	02/13/17	02/15/17	Not Started
Browse answers	4d	02/17/17	02/21/17	Not Started
View answers	1w	02/21/17	02/28/17	Not Started
Save answers	3d	02/23/17	02/26/17	Not Started
Update profile	3d	02/27/17	03/02/17	Not Started
Final testing, anything else not fin		03/03/17	03/09/17	
Project Deadline	1d	03/10/17	03/10/17	



Task Name	Duration	Start	Finish	Status
Proposal	1d	10/27/16	10/27/16	Complete
Presentation	2d	11/04/16	11/07/16	Complete
Functional Specs	9d	11/21/16	12/01/16	In Progress
Exams	7d	01/12/17	01/20/17	Not Started
Design Database	1d	01/30/17	01/30/17	Not Started
Login / Register	4d	02/01/17	02/04/17	Not Started
Android HTTP Client / API	3d	02/06/17	02/08/17	Not Started
Home screen & bottom nav bar	3d	02/09/17	02/12/17	Not Started
Server API	2d	02/13/17	02/15/17	Not Started
Browse answers	4d	02/17/17	02/21/17	Not Started
View answers	1w	02/21/17	02/28/17	Not Started
Save answers	3d	02/23/17	02/26/17	Not Started
Update profile	3d	02/27/17	03/02/17	Not Started
Final testing, anything else not fin		03/03/17	03/09/17	
Project Deadline	1d	03/10/17	03/10/17