Research Grant and Fund Management System

Web-Based Research Department Data Management System

Team 49

Allen Wang **Stylianos Rousoglou**

COMP103P Object-Oriented Programming

April 26, 2017

Department of Computer Science
University College London

Abstract

The Research Grant and Fund Management System is a web-based database application designed for the University College London's Research Services Department. It was developed by team members Allen Wang and Stylianos Rousoglou as a client project for the module COMP 103P - Object-Oriented Programming. The team worked in tandem with UCL Assistant Director of Research Services, Mark Burgess.

Mr. Burgess required a new system to manage the Research Service Department's information on individual students and research award allocations. The old management system was both confusing and complicated; all the research award information on allocations and students were stored in hundreds of excel sheets lacking data normality. Managing all this information was both slow and prone to data-integrity errors. The main issues that needed to be addressed were improving efficiency in accessing data, adding data, and maintaining data.

The team moved away from the cumbersome realm of excel sheets and implemented a web application that migrated data from excel into a database management system. The technologies used were HTML and JavaScript on the client side, NodeJS on the server side, and MySQL for the database. The front-end was implemented with consideration to client specifications, while the server and database schema were designed and developed entirely by the team members.

Since the long-term vision of the client requires UCL systems integration for user access and security reasons, the team focused primarily on developing a working proof of concept for the client. With a friendly user-interface and all functional requirements satisfied, the completed project allows the client to demonstrate the benefits of such a system in hopes of receiving the necessary resources and approval to move onto full integration with UCL systems.

Table of Contents

List of Tables and Figures

1 Introduction

1.1 Background and Problem Statement

1.1.1 Client and project background

University College London is one of the world's premier research institutions. Faculty and PhD students undertake various research projects that push the boundaries of medicine, life sciences, computer science, and the humanities [1]. A large portion of UCL's research is funded by the Research Councils UK (RCUK). RCUK is a strategic partnership of the UK's seven Research Councils and UCL is one of its primary beneficiaries in receiving funding awards [2]. These awards are broken down into one or many project allocations for the duration of the award. Since UCL has over 270 million pounds in RCUK funding, tracking the partitioning of an award among projects and its students and faculty supervisors is imperative. Research Services must know where each award is being used, and when applicable, make use of available extra funding to allow for more projects to be funded. Many years ago, the RCUK managed the information on all its research grants and allocations. However, as the funding amount increased and grew too large for one organization to handle, the responsibility of managing the research grant information was passed on to the individual recipient institutions, such as UCL. Given the large magnitude and importance of the work of UCL Research Services, it is both archaic and inefficient to store all the information in hundreds of excel sheets.

The data management difficulties of having to sift through these sheets to simply find or add records was the primary reason Mr. Burgess requested a better solution from our team. Another impetus was being able to promote more research opportunities for various departments; under the previous data management system, Mr. Burgess and his department would have to take in all requests for whether there was available funding. He desired a transparent system where individual UCL departments could view all the awards and allocations, thus giving them all the information on whether proposed projects could be funded. This would eliminate the possibility of planning a project but consequently finding out from Mr. Burgess that there were insufficient funds. He also envisioned more interdisciplinary projects flourishing as multiple departments could view the allocation of research grant funding, and propose joint cross-department projects and open new avenues for discovery.

1.1.2 Outline of project

The Research Grant and Fund Management System is a web based information management system. Its primary objective was aimed at improving the work flow and efficiency for the client, Mr. Burgess, and his colleagues (defined as primary users) in managing the large amount of data pertaining to UCL's research grants and project information. An integral part of this is the front-end web user interface; the primary users can search for records pertaining to awards, allocations, students, or collaborators. Since each entry of one category is related to other categories, the web application provides a detailed view of each entry that also has quick-links to related queries. Consequently, a primary user viewing a detailed allocation is only one click away from querying for all the students associated with that project allocation. These improvements to the primary user workflow are all supplemental to the main increase in efficiency: having all the excel sheets stored into a database that is maintained by a NodeJS server. Eliminating the cumbersome process of looking at individual excel sheets streamlines the look-up time greatly for the client.

The capabilities listed above are also available for users from outside the client's research department. These secondary users would be faculty and students looking to propose potential projects. Providing them a transparent view of the available funding and current awards satisfies the client's long term vision of fully utilizing all available funds by promoting research opportunities.
Adding, removing, and updating entries in the database are capabilities restricted from secondary users but available to primary users. For the project's purposes, a toggle button was used to switch between these two privileges. The user interface for secondary users simply hides the fields for adding, removing, and updating. Moving forwards, integrating with UCL users and granting permissions based on UCL identification would simply replace the current project's toggle button; users could be added to a primary user list and granted access to these features.

1.2 Approach to Project

Both team members worked tirelessly and efficiently to produce a functioning proof of concept in a condensed timeframe to overcome a 2-week initial delay in pairing with the client and receiving the initial project briefing. The first weeks of development consisted of client meetings to gather requirements as well as design a solution. After deciding on a database web application, the team focused on research to determine the technologies needed. During this phase, the team went through iterative database design as well as rapid learning of new technologies. After migrating excel sheets into the database, the back-end server was developed to manage the database while the front end

was created based off client feedback on user interface sketches. Although the front end and back end were developed by different developers, they were very much engineered in tandem because of many use cases and requirements involved multiple communications between the front and back end. This integrated development resulted in a web based application satisfying the client's requirements.

1.3 The Development Team

1.3.1 Background of Team Members

The team consisted of two members: Allen Wang and Stylianos Rousoglou. Both were third-year computer science students from the United States studying at UCL for the spring 2017 term as affiliate students.

Stylianos, known to his peers as Stelios, has an avid passion for computer science, having previously served as a course instructor and worked as a NodeJS developer. His familiarity with NodeJS made him an excellent and efficient back end developer for the project. He is originally from Athens, Greece and is an avid fan of Coldplay and The Killers.

Allen started his university studies far from computer science. Two years later, he is glad to have switched into a challenging and rewarding field. He previously worked as a web developer and this experience helped him in his role as a front-end developer for this project. He hails from Michigan, the motor-city area and is a big football fan.

1.3.2 Individual Team Member Roles

Both team members contributed to all facets of the project. Despite Allen being the front-end developer and Stelios being responsible for the back-end, they both worked with each other to give feedback and contribute in all areas of the project. The general responsibilities were outlined as follows:

Name	Roles
Allen Wang	UI Designer, Front End Developer, Client Liaison, Tester
Stylianos Rousoglou	Back-End Developer, Researcher, Report Editor, Tester

Table 1.1 – Team Roles

2 Requirements

2.1 Gathering Requirements

The first few weeks consisted of multiple meetings between the team and client to review and revise requirements. The feasibility and constraints of such requirements naturally formed as the scope of the project was finalized. With the feedback of the client, the team could categorize the requirements based on importance. User empathy played a vital role in the process, as the team was brought into the client's office and went through the workflow of the previous system to fully understand the limitations in fulfilling the various and necessary use-cases. The biggest necessity was a faster way of looking up information and a cleaner data management system.

The client and team came up with four "external portfolio views" to categorize the database information queries into: awards, allocations, students, and collaborators. By following this principle, the client also had a template to normalize the existing excel sheets, in which multiple awards could have different data columns. By cleaning up the data as well as further understanding the areas of improvement to the primary user workflow, the team aligned the requirements as closely as possible with the client's vision.

2.2 Persona

As previously mentioned, the application has two different types of users. The first, known as the primary user, consists of the client and his colleagues in UCL Research Services. They needed to be able to perform all possible operations on the migrated research grant data. These include conducting search queries, adding new entries, updating fields in existing entries, and removing entries.

The secondary user consists of faculty and students from departments within UCL seeking to view the funding distribution. They could be looking for available funding to use towards a new proposed project.

2.3 MoSCoW Requirements

The following requirements were the result of client feedback and approval. The first contributor listed was responsible for the 50% or more of the requirement implementation.

ID	Description	Priority	State	Contributors
1	The new system is to manage the information from the current system's hundreds of spreadsheets.	Must	Complete	Stelios, Allen

	After migration into a database, the system must also support adding, updating, and removing entries.			
2	Users can conduct different searches using different parameters to obtain a result-set for award, allocation, student, and collaborator information.	Must	Complete	Allen, Stelios
3	Client department users can update a notes section for each award, allocation, student, and collaborator row, providing a form of version control and logging/consistency.	Must	Complete	Allen, Stelios
4	Completed grants should be still recorded and searchable, per UCL guidelines in retaining records	Should	Complete	Stelios, Allen
5	Users have different privileges, managed by the client's department. Primary users have read/write access, secondary users only allowed to read.	Should	Complete	Allen, Stelios
6	For improved workflow, detailed results for data entries should have hyperlinks to related queries of different search categories (award -> allocation -> student -> collaborator).	Could	Complete	Allen, Stelios
6	The system supports integration with portico/UCL since UCL owns the information - data integrity	Won't	N/A	N/A
7	The system currently supports all of the RCUK grants, and eventually collaborators from external parties will be added to the system.	Won't	N/A	N/A

Table 2.1 – MoSCoW Requirements

2.4 Use Cases

The following lists the use cases developed by the team and client, with primary or secondary-specific cases noted.

ID	Description	User Note
1	Add research grant data	Primary
2	Update existing data	Primary
3	Remove data	Primary
4	Search for awards, allocations, students, and collaborators	Both

5	View a detailed entry from a list of query results	Both
6	Click to a related query search from a detailed entry view	Both

Table 2.2 – Use Cases

3 Research

Most the first few weeks of the project focused on research. Both Stelios and Allen looked at various frameworks, technologies, and strategies to implement the application. Similar projects were examined to find similarities and useful guidance. The entire stack of development was also critically examined as the team considered multiple options and weighed the costs and benefits of each individual component in the stack and how it fit into the entire application's development.

3.1 Related Projects

The team considered two of the most highly rated software systems online that seemed to address the client's needs. A cost-benefit analysis was conducted and several features from each solution were considered and used as inspiration for the eventual application.

Name	Description	Benefits	Drawbacks	Evaluation
Traverse by Kaseya	"Traverse proactively identifies data center and networking issues before they impact service levels, while providing you the flexibility to customize the system for your particular business needs."	Free Trial Unmatched horizontal scaling Cloud computing Data visualization Integrated database optimization for bottlenecks	Costs money after the free trial Abundant features surplus to core client requirements	Although this software is extremely powerful and contains most of the functionality needed, it does not make sense to spend resources for a subscription when the core functionalities needed are basic enough to implement.

Teamdesk	Custom database creation and management software for all users of all technical backgrounds.	Flexible custom database creation Secure cloud servers 99.96% uptime Unlimited storage space	No mention of visualization No demo or free trial period	A lower level solution compared to Kaseya, this software is closer to the client's goal but having to pay for a proof of concept is not ideal

Table 3.1 – Related Projects Analysis

3.2 Front-End

The team initially considered using popular frameworks like React or Angular for the client-side interface.

3.2.1 React

For React, the team was intrigued at learning this new technology and gaining experience with its component-based interface design. As well as being able to perform on the front end, React could also operate on the back end[3]. However, what ultimately prevented the team from using React was its role as only being a view layer, and its relatively steep learning curve [4]. The team's front end developer had limited JavaScript experience, and with a shortened timeline due to the delayed client-team matching, the team did not think the extensive time needed for ramping up on React would be worth its improved UI experience. As the client and team concluded, the most important thing was functionality in creating a data management system, with a clean user interface sufficing.

3.2.2 Angular

As for Angular, it would have provided benefits like those of React. It is a highly modular framework and would promote rich client-side interaction, as well as being more than just a view layer like React [5]. Angular is better equipped for serving as a Model-View Controller (MVC) which was crucial for the application since communication between the client and server was frequent [6]. However, the need to support frequent client-server interaction ultimately led to rejecting Angular simply because of the difficulty of understanding Angular code for unfamiliar developers.

3.2.3 Front-End Choice

Since the front-end and back-end development was very integrated, it would have proved a challenge for the back-end developer to understand the Angular front-end. It was then decided to use vanilla JavaScript with jQuery when necessary.

3.3 Back-End

Whereas the team approached front-end technologies with less familiarity, Stelios' previous experience with NodeJS made the back-end choice a lot clearer. Nevertheless, the team met with UCL's IT department to factor in the long-term implementation and scalability into the current choice of technology. The limited debate was between NodeJS and Java as the server-side platform.

3.3.1 Java

Using Java presented a set of benefits that were initially unexpected. The module for which this project was built towards had already presented Java to the team members, so familiarity was a plus. Upon meeting with UCL IT services, the team was informed that Java as a server-side technology would integrate seamlessly into using UCL services and authentication later. However, this would restrict the team to using UCL's older technology stack and limit its database and front end options. Therefore, Java was quickly ruled out as a contender.

3.3.2 NodeJS

Node's widespread use and extensive array of open source libraries made it the ideal choice. Along with Stelios' previous experience as a Node developer, Node also offered an easy mysql API library, Express for essential web application features, and the added benefit of being a lightweight server with the key feature of asynchronous communication. Allen also found it easier to pick up on Node's core features, such as promises and callbacks, making collaboration and feedback in the back-end environment smooth and efficient.

3.3.3 Back-End Choice

Unlike the deliberation of front-end technologies, where each had both advantages and slight disadvantages, on the back-end it was clear that NodeJS would give the team the most flexibility. Adding to the fact that the client desired a working proof of concept in which long term compatibility with UCL was not the most immediate concern, using Java could not compare to the efficient and lightweight asynchronous I/O of a NodeJS server. Node's mysql package also made it ideal since MySQL was the clear-cut choice as the database for the application.

4 Design and Implementation

4.1 Design

The user interface, system architecture, and overall application structure was designed with continued emphasis on the requirements and use cases for the primary and secondary users.

4.1.1 User Interface

The User Interface was sketched out with a simple template on Powerpoint. The interactive feedback and iteration between client and team led to developing a simple navigation bar that led to the four different query types.

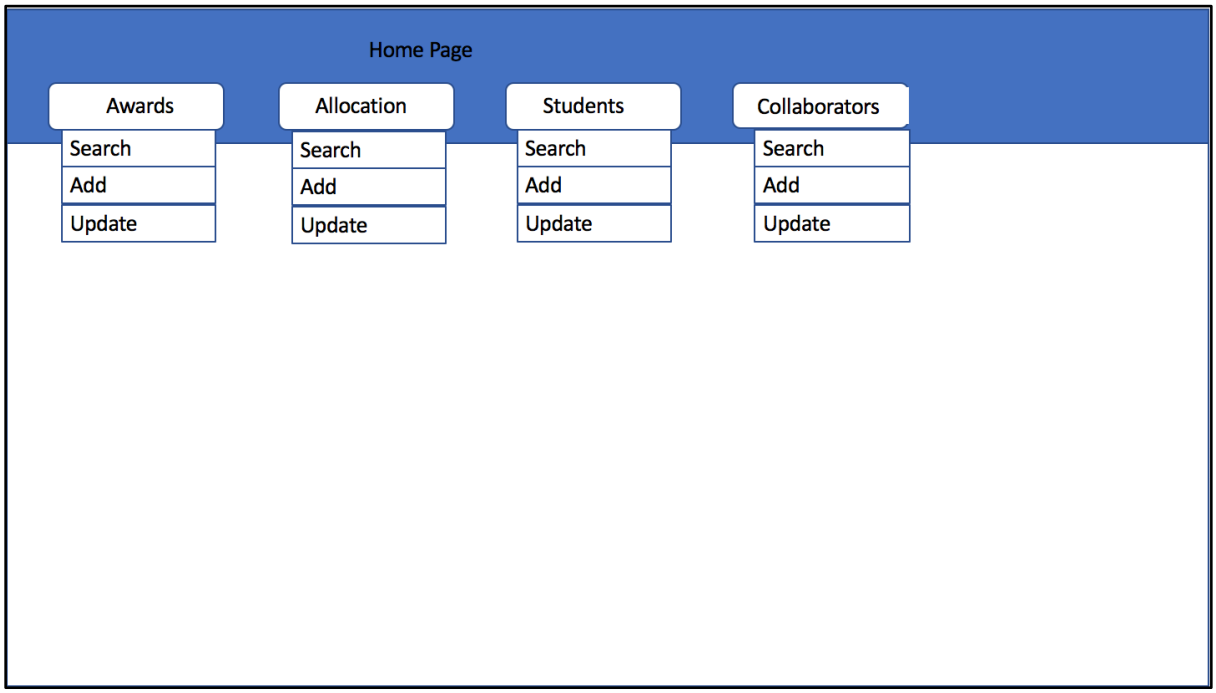


Figure 4.1 – UI Wireframe

At the same time, the team designed extra enhancements to the User Interface that were well accepted by the client. These include a dynamic detailed view of a data entry, with certain fields containing hyperlinks to the related queries pertaining to this certain data entry.

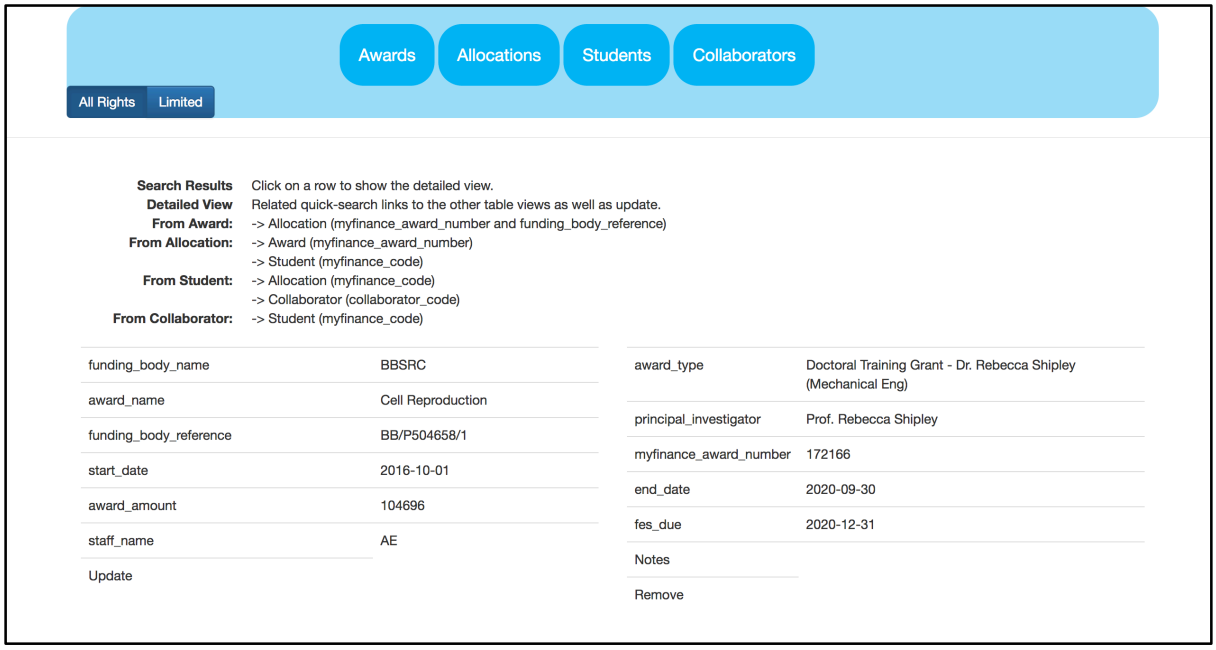


Figure 4.2 – Detailed View

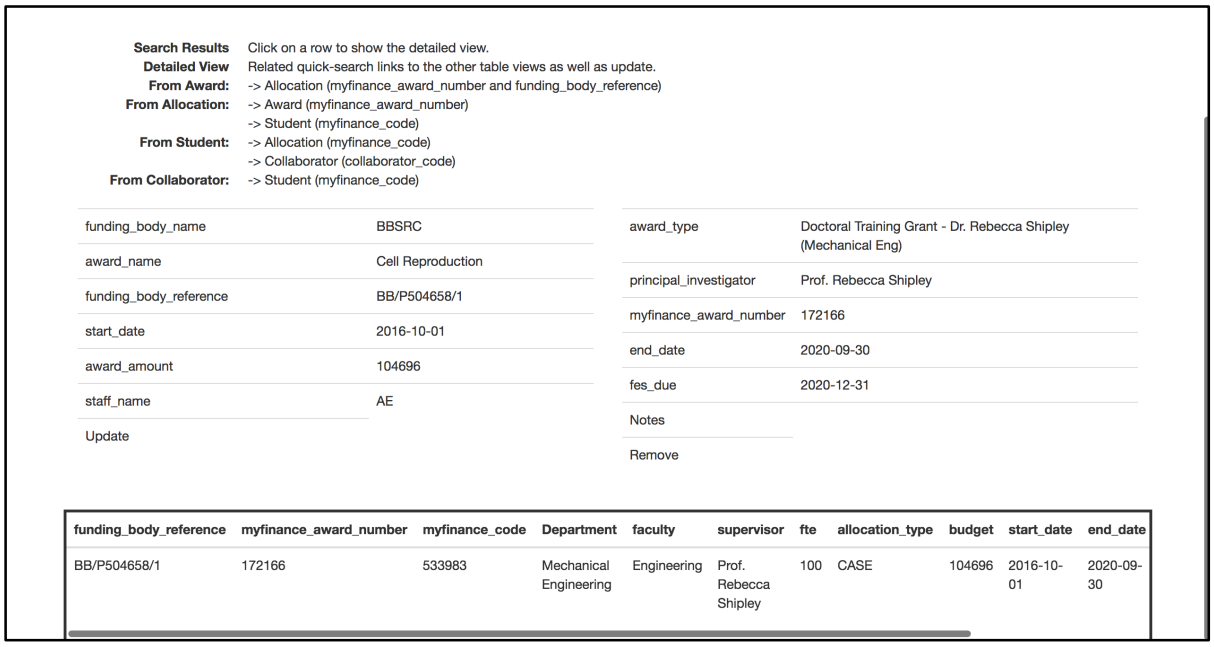


Figure 4.3 – Related Query Result from Detailed View

4.1.2 System Architecture

The system consisted of several components that enable the users to interact and fulfill the use cases required. Below is a diagram of the main components of the system and the actions that connect them to one another.

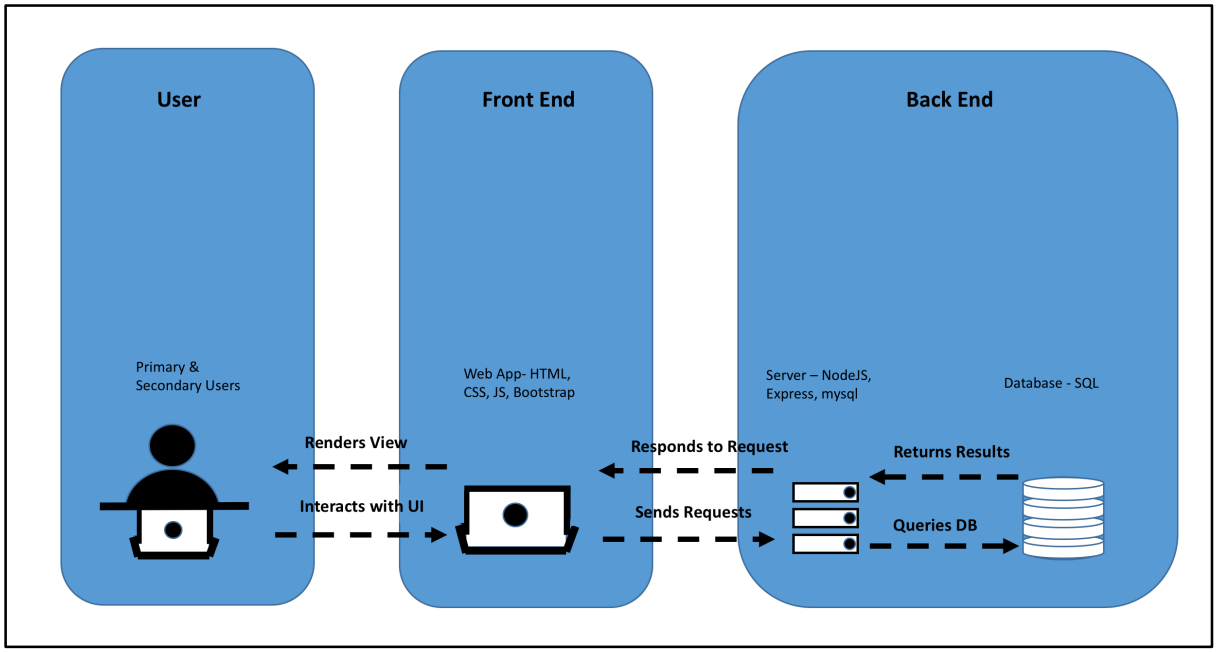


Figure 4.4 – System Architecture Overview

Front End

The front end is where the user is presented with a simple navigation bar linking to the four query types. Primary users will see a dropdown menu that allows them to add records to the database, while secondary users will only be able to search for records. This is where the users interact with the application; the front end takes in the user's actions then passes and receives information from the back end, presenting the resulting information to the user. The mechanisms behind communication with the back end and other functionality implementations are abstracted and hidden from the user.

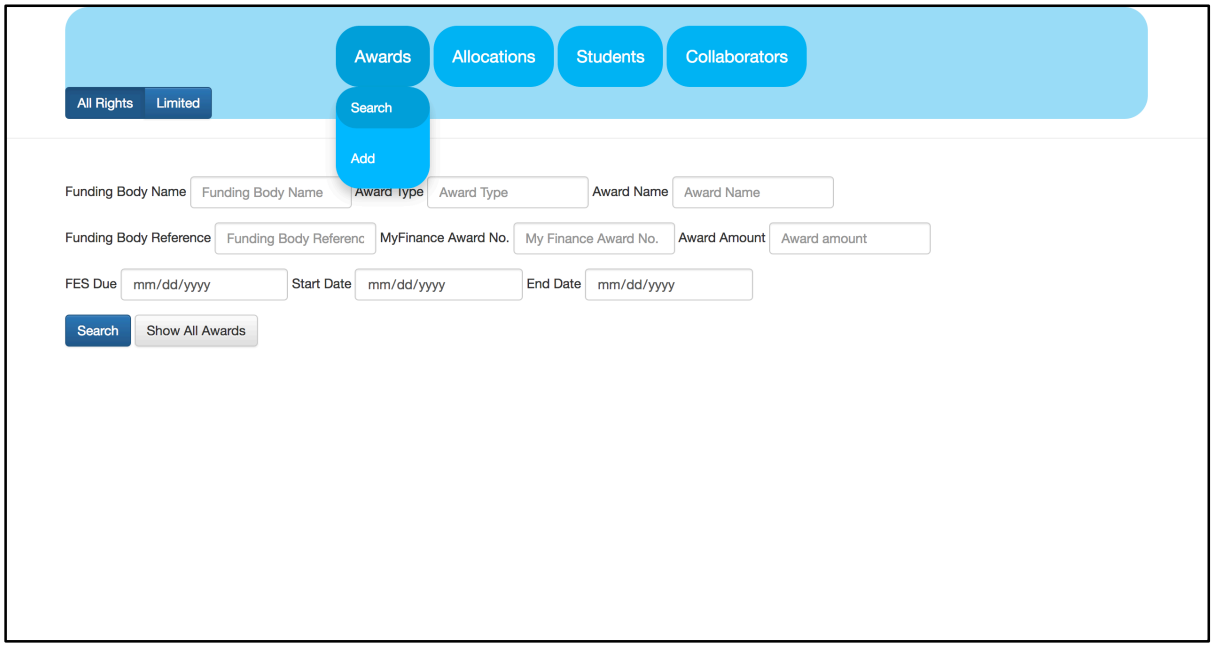


Figure 4.5 - Front End Search Form

Back End

The back end consists of a server that communicates with both the front end and the database. Requests are made to the server from the front end and consequently from the user interacting with the application. These requests trigger specific actions from the back end that can involve fetching information from the database and serving it back to the front end to present to the user. Some preprocessing is done in the back end on the information before responding back to the front end. The logic and complexity of building the server is hidden from the user; only the user interface interacts directly with the user and the front end and back end respond accordingly.

Database

The database entity relationship diagram was mapped out to contain the entries from the four different types of data sheets. Four tables were used for awards, allocations, students, and collaborators. Each table contains rows and columns, with the column values specifically describing the entry of the corresponding row. The relationships between the tables were noted based on primary and foreign keys that were present in records from different tables. Noting these relationships maintained data integrity in the event of removing one data entry and its related entries in other tables. The design of the database was integral in improving the speed of looking up information on all aspects of a research grant. The related queries available from the detailed view also made involved querying the database. During the requirements gathering phase, the team also went through the columns for each of the possible data values in each table to define the data types necessary for the database. Below is the completed entity relationship diagram.

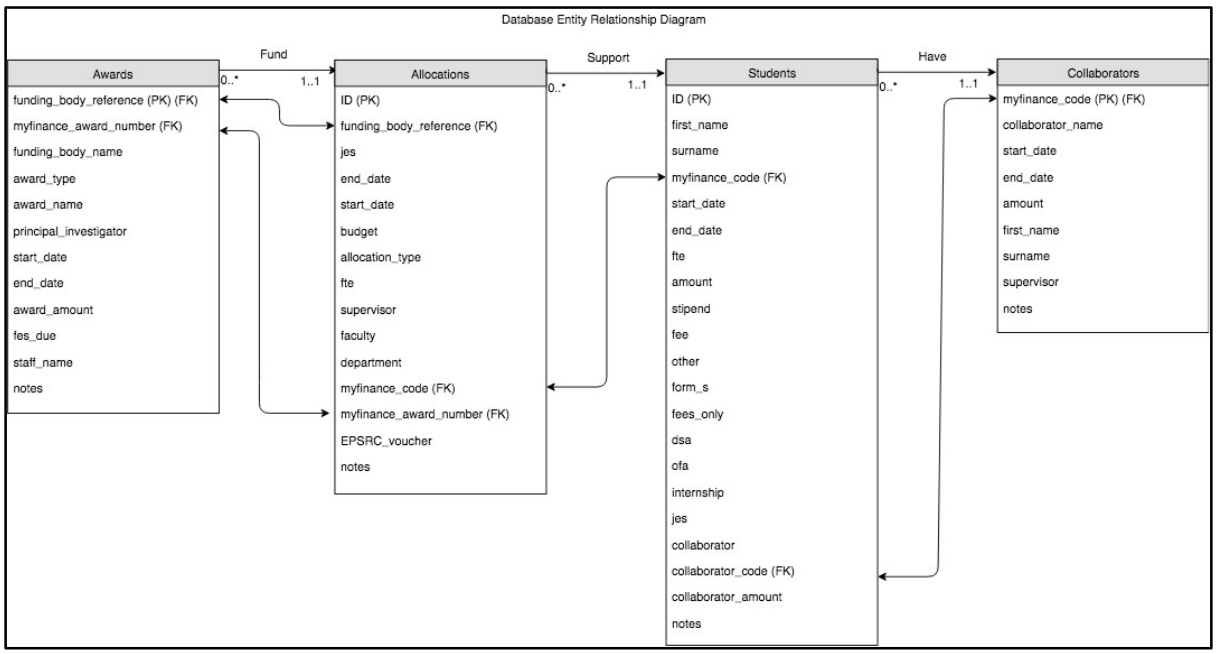


Figure 4.6 – Database Entity Relationship Diagram

4.1.3 Site Map

The different pages and views of the application are show in the site map below. The application structure is like the site map except with four different query types for each site view.

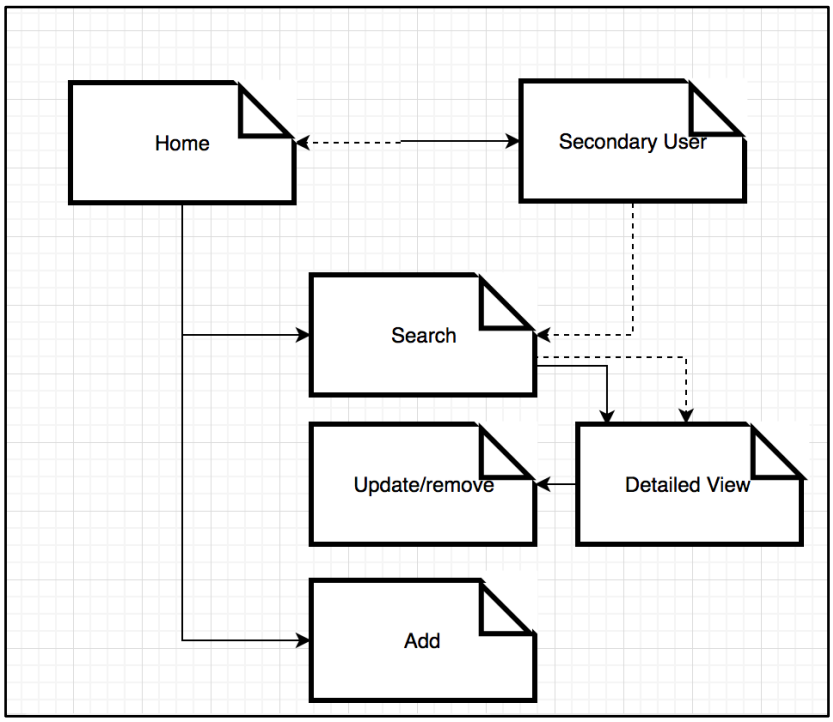


Figure 4.7 – Site Map

4.1.4 Application Structure

The application is structured so that the users have multiple quick and easy methods of accessing certain records in the database. If the specific fields of an entry are known, the user can input the fields known and select the desired record to display. Alternatively, using the hyperlinks for related queries, a user can navigate to any desired record in the system in around five clicks even if the user does not remember any of the search fields for the desired record. Below is a simple diagram of a user's possible interactions with the application. The dashed lines represent secondary user interaction and the solid lines represent primary user interaction.

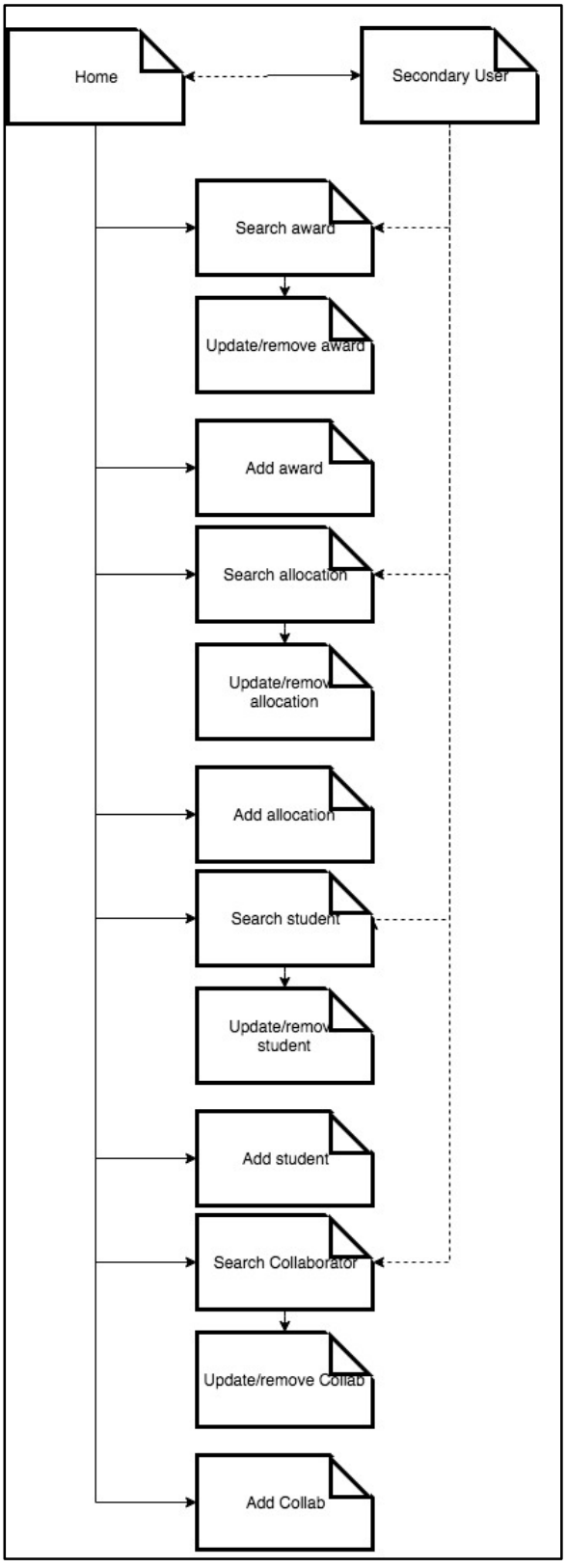


Figure 4.8 - Application Structural Flow 1

4.1.5 Design Patterns

A design pattern is a widely accepted and reusable type of solution to a problem that is quite common. The team implemented several design patterns during the development of the application, namely the Model-View-Controller, Command, Observer, Module, and Lazy Initialization design patterns.

Model-View-Controller (MVC)

MVC, one of the most well-known patterns, and the basis for many GUI frameworks, was used to design the application. Generally regarded as an architectural level pattern, MVC splits the design process into three areas: data, event handling, and visible representation. The model component holds the data and triggers the view component upon data changes. The view component receives data and represents it for the user, while also notifying the controller when views change state. Finally, the controller handles events and chooses the correct model or updates the view. The different components of the application contribute to these different needs. In terms of user interaction, the

user sees the view and interacts by sending inputs to the controller. A diagram of the MVC model is shown below.

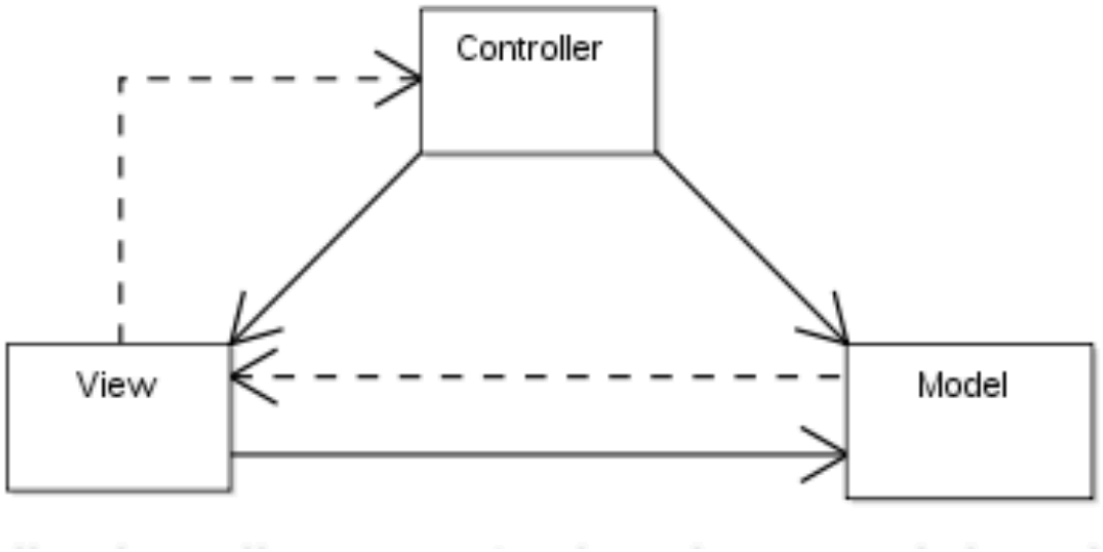


Figure 4.9 - Model View Controller Diagram

Command

The command design pattern encapsulates requests as if they are objects, primarily to execute or wait on requests at different specific times. This establishes a history of requests and resembles the control aspect of threading and asynchronous I/O. The team utilized concepts of the command design pattern in the back-end server implementation primarily with promises and resolve/reject call backs.

Observer

The observer design pattern is characterized by different components in the application having a one to many relationships. This dictates communication between components, and a single change causes those dependent elements to automatically update as well. The observer design pattern is integrated into MVC, as the view component inherently observes and waits on the model component. An example in the application is how clicking on a detailed view for a record initiates and renders the update button, remove button, and related hyperlinks. These components wait for the detailed result to be selected to select of use the correct data.

Module

The module design pattern partitions related code into work packages that serve more specific functions. This allows for modularity, faster development, and efficient code organization and maintenance. In this application, modules flourished in both the front and back end. The front end contained modules that

handled searching/adding records and removing/updating records. They were distinct because removal and update options observe only are presented when a record is rendered in a detailed view. By separating these two, development could proceed in parallel and once both were completed, the modules seamlessly interacted. On the back end, the core server component handled requests from the client by utilizing other specific modules. These include a module for database operations and the HTTP request module.

Lazy Initialization

Lazy initialization solves the problem of slow bottlenecking by only performing certain operations when they are needed. This "lazy" approach ensures that whichever process needs the most resources at any given moment should theoretically have access to the resources since other processes will not be taking up the computer's resources until they require it. One example is the update form, which only is rendered when the update "onClick" event handler is triggered. The form is then populated with the current values of the record. This process takes up the computer and application's resources only when it is necessary.

4.2 Implementation

4.2.1 Development Tools

The team utilized several development tools that maximized efficiency to aid in completing this project. An online integrated development environment (IDE) was chosen primarily because constantly testing the application live was important. Cloud9 offered a free web server hosting service and this streamlined the development process tremendously. Instead of working on local branches, all branch development was on the cloud. Since the application quickly reached the phase where online deployment was crucial for testing, using Cloud9 proved to be an ideal choice. When online access was limited, Allen would use Sublime to work on the front end code while testing on his local machine.

The Cloud9 IDE was also integrated with GitHub. GitHub is a version control tool that manages the main repository for a project. Contributors can branch off while working on different features, then merge their changes back pending an approved pull request. The team utilized Cloud9's ability to synchronize with their GitHub repository to ensure that their rapid testing of the application still followed the standard procedures of version control and collaboration through GitHub.

A Slack team chat channel was also created. Slack is a communication tool where team members update the rest of the team on progress or other important development issues.

The development tools used in this project and their roles are shown in the diagram below.

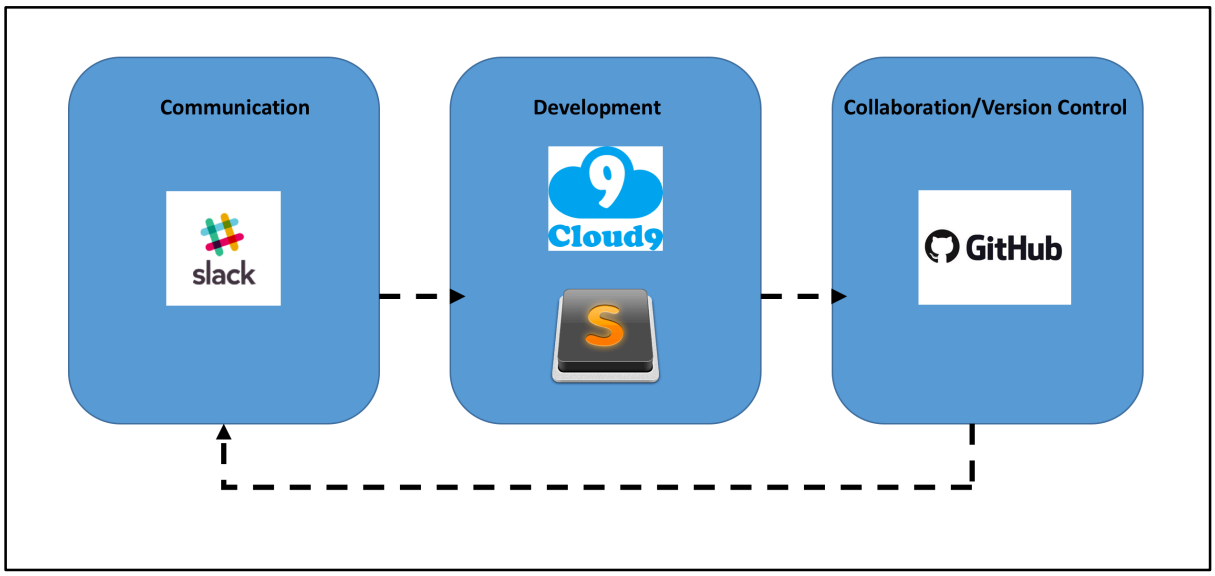


Figure 4.10 - Development Tools Diagram

4.2.2 Front End Implementation

Like most common web applications, the front end of this project was built with HTML + CSS + JavaScript. Additional frameworks were also used for responsiveness and dynamic views.

In constructing the layout of the web application, Bootstrap was integrated due to its clean form styling and responsiveness for multiple platforms. Bootstrap is a CSS web development framework that offers clean and easy styling of HTML elements.

As previously discussed, the client side view was built using vanilla JavaScript, rather than a popular framework like React or Angular. The dynamic web application utilized several traditional elements of JavaScript like event handlers and DOM manipulation. Using just JavaScript gave the developers more control over the front end and increased their understanding of JavaScript without simplifications offered by frameworks.

4.2.3 Back End Implementation

The popular NodeJS runtime paired with Express web server to create a fast and lightweight server. Using the node package manager, the team installed the node mysql library to manage the SQL database. This framework provided abstracted methods to manage the database and ensured the entire back-end development utilized JavaScript. Express endpoints were the avenues for communication with front end requests. The server would then process these requests before accessing or modifying the database with the mysql package. Since the front and back end both communicated with HTTP requests passing JSON objects, the mysql extension was extremely useful as it also passed information from the database in JSON format.

4.2.4 Key Functionality Implementation

The following sections detail the implementation of the core "must-have" requirements for the project, from a full-stack perspective.

Add Data to Database (via Excel sheet migration or form input)
Initially, the client's data was spread over hundreds of Microsoft Excel sheets. Specifically, for any given award, at least two spreadsheets would be required to hold all pertinent information. Therefore, prior to data migration, there was a need for some data manipulation to condense information in an organized fashion. Mark, our client, also helped in this process, by providing some sample data in a single spreadsheet, formatted consistently (unlike data in spreadsheets that was inconsistent and occasionally partial.) With the data at hand, the database tables were constructed with SQL command-line queries and tailored to the data fields Mark wished to retain. Data migration then was a matter of running a short NodeJS script to import the csv data file into the database.

Search for Different Record Types

Under the client's previous data management system, a loose file organization system using folders and Excel sheets meant that a lot of familiarity with the relative location of certain records was necessary to look up any record with any reasonable speed. Even if a new user knew all the column values in a record, without prior knowledge of the folder organization of the hundreds of Excel sheets, searching for records would be extremely tedious.

The team's solution inherently improved this functionality greatly by storing records in a database with four tables. The front end presents users with a navigation bar to the four different search forms where the user can search the specific table, whether it be award, allocation, student, or collaborator. The client side JavaScript then processed the user inputs for these four search forms and

would make a fetch HTTP request to an Express endpoint on the back-end server. The back-end then queried the database, then returned the response to the client either with a result set or null set. The front-end then rendered the results in a view for the user with the ability to click each record and view it in greater detail. Thanks to the design patterns mentioned previously, this functionality performed a lot more efficiently in the application compared to the previous system.

Update and Remove Records

The team could use modularity to develop the update and remove functionalities in parallel with searching functionality. The team decided that a user should be presented the options of updating or removing only after seeing the detailed view of a record. Since the detailed view of a record was presented upon clicking a search result, there would have been a bottleneck in development waiting to finish implementing search before starting update and remove. Upon rendering the detailed view of a record, the client generates two buttons with JavaScript that both contain the current record's columns. These values are used as the parameters to either update or remove the correct record.

If update is chosen, the update form is rendered with the record's previous values all prefilled into the form. A cancel option is also present which takes the user back to the detailed view. After the user makes changes to the form and submits, the front end sends the two form inputs to the back end; the first input contains the values of the record to change and the second contains the desired new changes of the record. The back end then runs an update query on the database using the first input as a parameter and the second as the update field.

For remove, a confirmation pop up is rendered before allowing the user to proceed. Once confirmed, the parameters of the original record are sent to the back end which executes a delete query.

Both functionalities return either a success or fail response to the front end. Should the access rights be toggled from "all rights" (primary users) to "limited" (secondary users) then the update and remove buttons are never rendered. This prevents secondary users from modifying the data in the system, effectively giving them read-only permission.

4.2.5 Data Storage (JSON) and database

JSON is the data structure we used to store data for the front end view component. Each row in the search results contained a hidden field with the record's values stored as JSON. The JSON was iterated to generate the detailed view for each record when clicked. The team chose to store the data instead of having to make an additional query to the back end and database whenever a

detailed view was clicked. This improved the responsiveness and speed of the application at the relatively low cost of using JSON. Saving the JSON data for each record also helped in speeding up the remove and update functionalities, since the front end made requests to the back end using JSON. These communications contained parameters and changes to the database in the form of JSON, thus storing each row with the column values as JSON parameters served multiple purposes that outweighed the cost of extra storage. In the back-end, whether adding, updating, or deleting a record, iterating over a JSON object with the necessary data fields was a very convenient way of generating the necessary SQL queries for communication with the database.

4.2.6 Package Tree

The following diagram details the structure of the application's development repository. Directories left unexpanded are the node modules and the fonts folder.

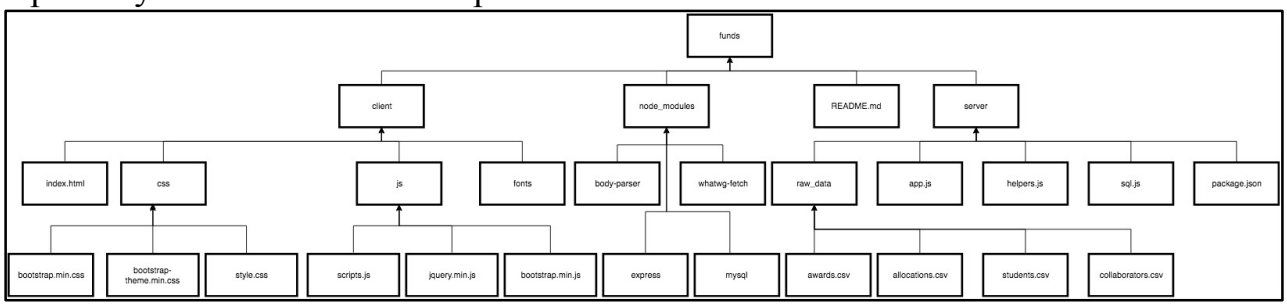


Figure 4.11 - Package Tree Structure

4.2.7 Project Management

To combat the delayed start to the project, the team made sure to schedule their deliverables in accordance with strict timelines to meet the deadline for their client. A month before the project was due, they already had basic functionalities working and performed a live demonstration for the module instructor and client. After starting behind schedule, within two weeks the team's progress had already caught up and eventually surpassed the expected progress timeline. This was achieved by a combination of hard work and efficient project management.

Gantt Chart

The following Gantt Chart shows the visualization of the project timeline. The timeframe for completing tasks that led to the completion of the project are presented using the bi-weekly reports for the module as a reference.

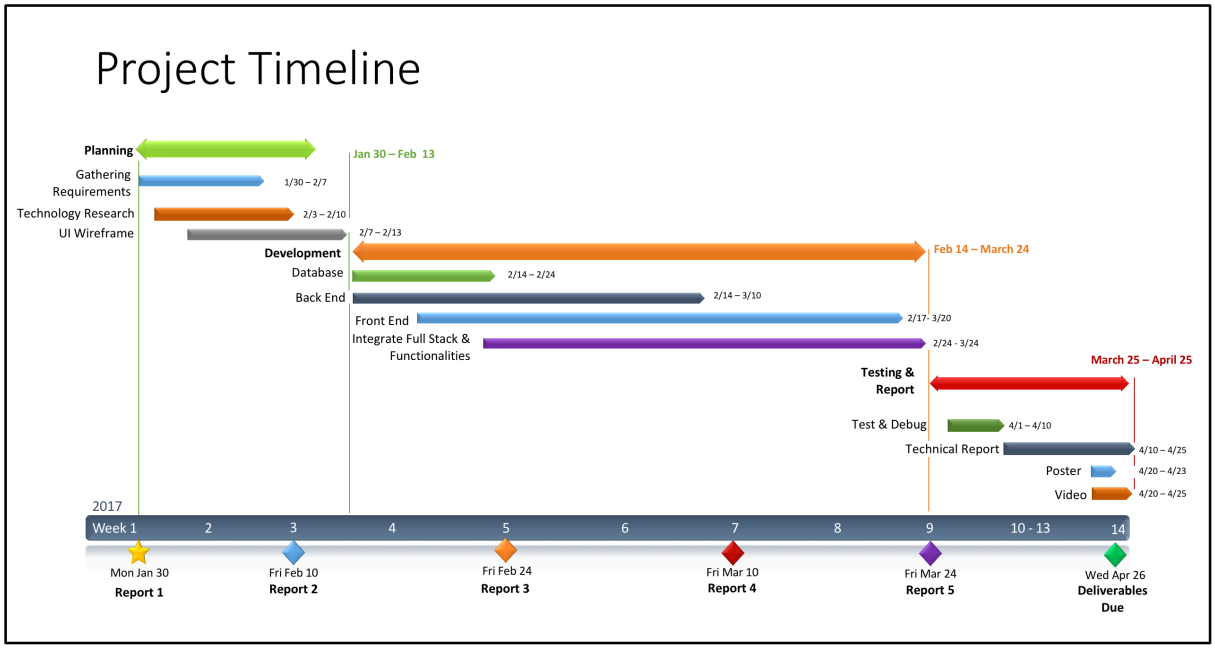


Figure 4.12 - Gantt Chart Project Distribution

5 Testing

5.1 Functional Testing

The process of testing our software solution was methodical and thorough. To begin with, the core code development was done in a highly modular fashion. This allowed for debugging specific functions and small parts of the code individually with designated tests, and helped avoid unexpected bugs later on. Early in the development process, the team devised an extensive list of corner and edge cases that would be used to test the robustness of any piece of code written. This list includes, but is not limited to, empty fields, numerical strings, strings containing symbols, and non-ASCII characters. We ensured that the program handles all the aforementioned cases gracefully, both during module testing and end-to-end tests. During the latter stages of the development process, we regularly tested our product as a whole, making use of all available functionality, and ensuring that the continuous development and addition to the code base did not introduce any unexpected software bugs or other issues

5.2 Compatibility Testing

Development of the product was primarily done using our preferred browser, Google Chrome. However, we learned why compatibility testing is crucial the hard way, when we tested our working solution on Safari and it was totally unresponsive. Nevertheless, it turns out that JavaScript compatibility with Safari is not an uncommon problem. As research revealed, *fetch,* a function extensively used in our front end code to make post requests to our server, is not native to the Safari browser, so the application was unable to communicate with our database. Fortunately, the issue was resolved by adding a *Javascript polyfill*[1] to our code base that restored the functionality of *fetch.*

The full results of our compatibility testing process can be found below.

Device & Browser	Comments	Results
Desktop (Chrome)	Mac, Linux, Windows	All testing successful
Desktop (Firefox)	Mac, Windows	All testing successful
Desktop (Safari)	Mac	All testing successful
iPhone 7 (Safari)		All testing successful
iPhone 7 (Chrome)		All testing successful
iPhone 6 (Chrome)		All testing successful
iPhone 6 (Safari)		All testing successful
iPhone 6 (Opera)		All testing successful
Android (Chrome)		All testing successful

[1] https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills

5.3 Responsive Design Testing

Although the product was designed as a web application to be used primarily on Mark's wide computer screen, it was important to ensure that the web application would be functional and responsive irrespective of the environment used to access it. Therefore, we repeated the testing process on a variety of different screens, including Macbook Air and Macbook Pro machines, desktop devices running the Windows OS, and popular smartphones such as iPhones 6 and 7 and Android 6. We ensured that the application is intuitive and easy to use on all devices, and adjusted our user interface slightly to accommodate the "fat-finger" problem potentially arising on mobile devices.

The full results of our responsive design testing process can be found below:

Device	Results
Macbook Pro 15-inch (2880 x 1800)	All testing successful
Macbook Air 11-inch (1440 x 900)	All testing successful
iPhone 7 (1920 x 1080)	All testing successful
iPhone 6 (1334 x 750)	All testing successful

5.4 User Acceptance Testing

In order to decide on the strong aspects of our product's user interface and on potential future improvements, we employed the help of friends and classmates. Due to the sensitive and confidential nature of the data at hand, we used dummy data throughout this phase of testing to protect the information Mark has entrusted us with.

Positive Feedback	• Intuitive user interface • Minimalist design • Thorough search options • Fast and responsive interface
Areas for improvements	• More environment-specific responses (e.g. automatic resizing for phone screens) (out of the scope of this project) • Explanation of data fields (redundant, given that potential users are already familiar with the nature of the data) • More aesthetically pleasing data display

6 Conclusion

6.1 Summary of Achievements

Requested Features

ID	Description	Priority	State	Contributors
1	The new system is to manage the information from the current system's hundreds of spreadsheets. After **migration** into a database, the system must store the data reliably and securely.	Must	Complete	Stelios, Allen
2	Adding, editing, and removing entries from tables should be intuitive and simple	Must	Complete	Stelios, Allen
3	Users can conduct different searches using different parameters to obtain a result-set for award, allocation, student, and collaborator information.	Must	Complete	Allen, Stelios
4	Client department users can update a notes section for each award, allocation, student, and collaborator row, providing a form of version control and logging/consistency.	Must	Complete	Allen, Stelios
5	Completed grants should be still recorded and searchable, per UCL guidelines in retaining records	Should	Complete	Stelios, Allen
6	Users have different privileges, managed by the client's department. Primary users have read/write access, secondary users only allowed to read.	Should	Complete	Allen, Stelios
7	For improved workflow, detailed results for data entries should have hyperlinks to related queries of different search categories (award -> allocation -> student -> collaborator).	Could	Complete	Allen, Stelios
8	The system supports integration with portico/UCL since UCL owns the information - data integrity	Could	Out of scope	N/A
9	The system currently supports all of the RCUK grants, and eventually collaborators from external parties will be added to the system.	Won't	Out of scope	N/A

Workload Distribution

ID	Work Package	Category	Contributors
1	Meeting Client to Gather Abstract Requirements	Client Interaction	All
2	Analyze and form MoSCoW requirements	Requirements Analysis	All
3	Formulate use cases and iterate and review with client	Client Interaction /Requirements Analysis	All
4	Research Possible Technologies and different Implementations	Research	All
5	Front End research	Research	Allen
6	Back End/Database research	Research	Stelios
7	User Interface Design	UI Design	Allen
8	Search Forms Rendering and Processing	Front End	Allen
9	Client side JS for dynamic rendering of results and communicating with server	Front End	Allen
10	Results quick-search hyperlinks within detailed view	Front End	Allen
11	Database Design	Back End	Stelios
12	Database Creation and Management	Back End	Stelios
13	Excel Data Migration	Back End	Stelios
14	Server receiving and handling queries to the Database and serving back to client	Back End	Stelios
15	Bi-Weekly Report #1	Reports	All
16	Bi-Weekly Report #2	Reports	All
17	Bi-Weekly Report #3	Reports	All
18	Bi-Weekly Report #4	Reports	All
19	Bi-Weekly Report #5	Reports	All
20	Technical Report Abstract	Reports	Allen
21	Ch. 1 Technical Report Introduction	Reports	Allen

22	Ch. 2 Technical Report Requirements	Reports	Allen
24	Ch. 3 Technical Report Research	Reports	Allen
25	Ch. 4 Technical Report Design and Implementation	Reports	All
26	Ch. 5 Technical Report Testing	Reports	Stelios
27	Ch. 6 Technical Report Conclusion and Future	Reports	Stelios
28	Compatibility and User Testing	Reports	Stelios
29	Technical Report Appendix	Reports	Allen
30	Technical Report References	Reports	Allen
31	Poster	Poster Design	Stelios
32	Video	Video Editing	All

Individual Contribution Table

Work Package	**Allen**	**Stelios**
Client Liaison	60%	40%
Requirement Analysis	50%	50%
Research	40%	60%
UI Design	75%	25%
Front End	80%	20%
Back End	10%	90%
Testing	25%	75%
Technical Report	66%	34%
Bi-Weekly Report	50%	50%
Poster Design	0%	100%

Video Editing	50%	50%
Overall Contribution	55%	45%
Roles	UI Designer, Front End Developer, Client Liaison, Tester, Report Editor	Back-End Developer, Researcher, Report Editor, Tester

6.2 Critical Evaluation

From our first meetings with Mark, we discussed extensively what the purpose of the project should be. We agreed that building the perfect tool for him would be far beyond the scope of this class assignment, as there are vast amounts of empowering features that could be implemented but would require a full-time commitment and professional work, access to UCL APIs, integration with Portico, etc. Mark highlighted that UCL has been unable to provide him with professional software for his work in the past, siting an inability to find such software that caters to his exact needs. After further discussions, we concluded that the best course of action would be the following: develop a web application prototype tailored to Mark's needs and wants, which he would use to demonstrate the benefits of a tailor-made professional software solution compared to the current system in place. This demonstration, he hoped, would expose his immediate need for a new professional solution and urge UCL to provide him with funds for one.

The final product is very satisfying overall. It's a lightweight, clearly written, NodeJS web application that uses the latest technology in front-end and back-end web application design, such as *asynchronous promises,* for speed and efficiency. The outcomes decided upon over the course of our meetings with Mark have been met, and the software solution closely emulates Mark's initial vision in terms of its functionality. The code is clearly documented and robustly tested, compatible with all major devices, browsers and operating systems, and provides an intuitive, easy-to-use interface. First, we developed a streamlined process for standardizing Mark's unordered and unformatted data, and importing the information into clearly defined and logically intuitive SQL tables. Apart from organizational advantages, the database integrity guarantees offer more reliability and security, and minimize the impact of possible human error on large volumes of sensitive information. More importantly, tasks that Mark repeats daily, which currently involve scrolling and searching through lists of Excel documents, are now a matter of a single click. Searching, adding, editing, and deleting information now involves querying the database via an intuitive and convenient platform that interacts with the data transparently and only presents the user with relevant results. Under our solution, Mark never has to interact with the entirety of the data stored. On top of that, such a web application approach allows him to

work remotely and not rely on his work computer, where the Excel spreadsheets are stored.

6.3 Future Work

Had we had more time to dedicate to this project, we might have followed a different course of action and set significantly different objectives for our end result. Instead of a proof of concept, we could have attempted to implement a tool that Mark could actually use in his work. However, such a tool would have to be taken over and maintained by the UCL IT department, which would require numerous meetings with UCL engineers, development of the product in completely different programming lagnuages (for compatibility with UCL engineers' work), and other time consuming processes that we decided were beyond the scope and timeframe of the class's project.

7 References

"Research Councils, UK" [Online]. Available: *http://www.rcuk.ac.uk/*. [Accessed: February 2017]

"Research services, UCL" [Online]. Available: *https://www.ucl.ac.uk/finance/fba-teams/research-services* [Accessed: February 2017]

"React Tutorials" [Online]. Available: *https://facebook.github.io/react/docs/hello-world.html.* [Accessed: February 2017]

"Advantages and disadvantages of React" [Online]. Available: *http://www.pro-tekconsulting.com/blog/advantages-disadvantages-of-react-js/*. [Accessed: April 2017]

"Angular JS" [Online]. Available: *https://github.com/angular.* [Accessed: February-April 2017]

"Advantages and disadvantages of AngularJS" [Online]. Available: *http://www.software-developer-india.com/advantages-and-disadvantages-of-angularjs/.* [Accessed: April 2017]

Appendix

A. User Manual

The user interface of the application is designed to be intuitive and easy to use. Depending on what data the user wants to access or modify, they can click Awards, Allocations, Students, or Collaborators. To add information to the database, one should click 'Add' on the corresponding drop-down menu, then proceed to filling in the provided form and hitting 'Add'. To search for data entries, one should hit 'Search' on the appropriate drop-down menu, fill in the information they wish to query with, and hit search. After the results appear, the user selects the entry of interest, which becomes the main view of the page, and is presented with two additional buttons: 'Update' and 'Delete'. Hitting 'Update' will present the user with a form that allows them to modify any data they wish, subsequently clicking 'Update' to commit and store the changes, or 'Cancel' to discard them. Hitting 'Delete' removes the chosen entry from the database.

B. Deployment Manual

In its current state, our application can only be ran from the environment it was developed in, namely the Cloud9 IDE (www.c9.io). The cloud9 environment is not only an extremely convenient platform, but also provides a built-in database server, which makes development and server-database integration easier. Since our system currently uses the database server running on our shared IDE's localhost, the application can only be ran from within a local terminal that has access to localhost, and only after the database server has been started.

As with every node application, NodeJS needs to be installed and updated in order for the app to be launched.[2] In addition to that, all library dependencies must also be installed. the *Node Package Manager,* or *npm,* takes care of that; after successfully installing node, executing '*npm install*' in the application's home directory (i.e. where *package.json* is located) will automatically install the latest versions of all dependencies listed in the package file.

On the Cloud9 IDE, the application can then be ran simply by executing the bash script '*test'* (included in the code submission), which simply starts the database server, and then runs the application's entry-point, namely *app.js*. This process can also be followed manually, by executing 'mysql-ctl restart', 'cd ~workspace/funds/server', and 'node app.js', in that order. If running the application off the cloud, modifying lines 19-26 of *app.js* is necessary to establish a connection with the database server used, but the first step described here is no longer necessary.

[2] https://nodejs.org/en/download/

C. System Manual

Standard NodeJS application development practices have been followed throughout the production of our solution. The entry-point of our application is named 'app.js', following the popular convention, and the *package.json* file contains all necessary compatibility information and dependencies to run the software.

The code is structured as follows: the *server/* directory contains the back-end entry-point, as well as two files with helper code functionality, namely *helpers.js* and *sql.js.* Whereas *app.js* loads all relevant libraries, connects to the database server, and defines all the active server endpoints, *sql.js* contains all functionality that renders and executes SQL queries. Finally, *helpers.js* contains some helper methods for string manipulation, numerical calculations etc.

The *client/* directory contains the main html view file, *index.html,* as well as three subdirectories, *css, fonts,* and *js,* where stylistic elements and front-end scripts reside. Specifically, *js/scripts.js* contains our front-end JavaScript code, whereas *bootstrap.min.js, jquery.min.js,* and *fetch.js* contain third-party code that supplements our functionality.

Lines 19-26 of *app.js* establish a connection to the SQL database. In developing and demoing the product, we used Cloud9's built-in database server, with *localhost* as the host and *3306* as the port (which is specific to Cloud9). If using a different database server (which will likely be the case), the parameters of the connection have to modified to match those of the server used.

D. Code Citation

No.	Function Name	File Name	Source
1	findPos	scripts.js	http://stackoverflow.com/questions/11880443/how-to-scroll-browser-to-desired-element-by-javascript
2	Number.prototype.format	scripts.js	http://stackoverflow.com/questions/149055/how-can-i-format-numbers-as-money-in-javascript
3	N/A	fetch.js	https://github.com/github/fetch/blob/master/fetch.js

E. Client Feedback

Dear Mark,

That sounds good. I also wanted to allow you to look over the UI template that I drew up based on your sketches. Of course we will hash out more of the details and complexities of information tomorrow, but if you wanted to take a look before our meeting that would be great.

Sincerely,

Allen

Hey Mark,

Just wanted to send you our bi-weekly report for you to check when you get back.

Cheers,

Allen

Allen

I think a video of the basic functionality would be a good start;

after that perhaps the user interface;

then reports;

and perhaps a video of the required UCL architecture to support the system.

This could then lead to an overall demonstration of the proof of concept.

Many thanks

Mark

Hey Mark,

Apologies but I have been unable to get adequate wifi to upload this video. Here is the basic functionality of the project, and I am still working on the final report that I will submit to both my instructors and you.

Sincerely,

Allen

https://www.youtube.com/watch?v=Pp34xOjsZAc

Alan

The basic functionality looked to be all there. Assuming that you can deliver a basic working model for me to use to sell the proof of concept and document as previously discussed I am happy with the work done.

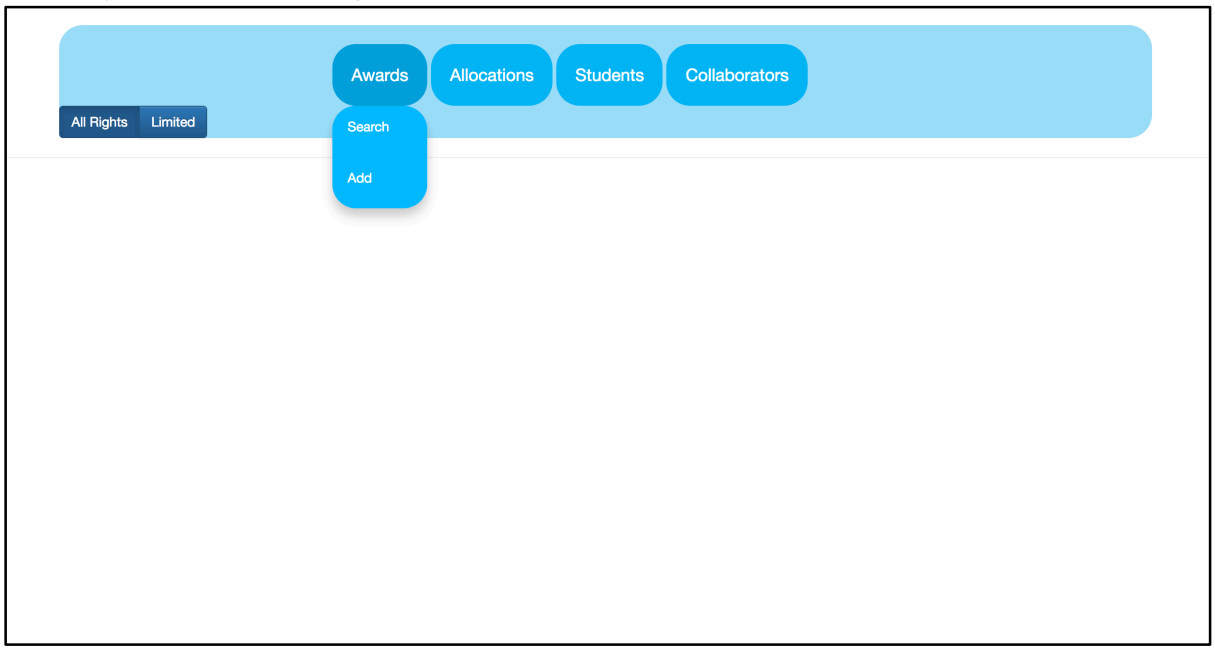
And thank you.

Mark

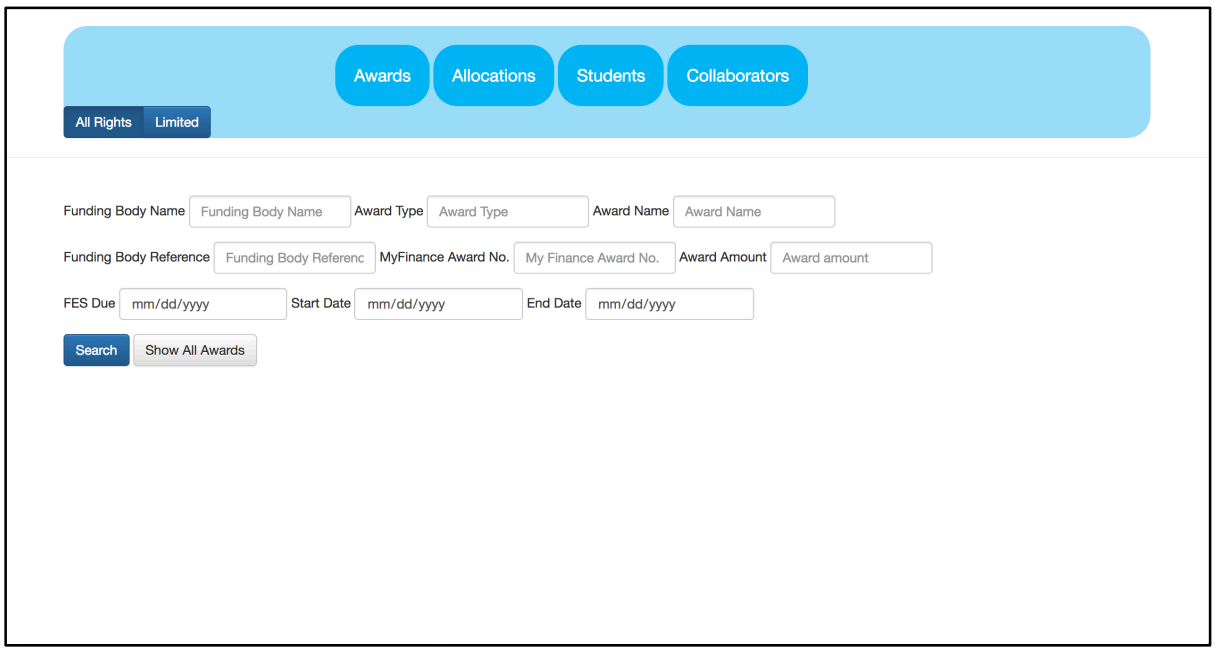
Sent from Samsung tablet

F. Key Screenshots

Primary User Home Page



Search Form



Search Query Results

Search Results Click on a row to show the detailed view.
Detailed View Related quick-search links to the other table views as well as update.
From Award: -> Allocation (myfinance_award_number and funding_body_reference)
From Allocation: -> Award (myfinance_award_number and funding_body_reference)
-> Student (myfinance_code)
From Student: -> Allocation (myfinance_code)
-> Collaborator (collaborator_code)
From Collaborator: -> Student (myfinance_code)

funding_body_name	award_type	award_name	principal_investigator	funding_body_reference	myfinance_award_number	start_date	end_date	award_amount
AHRC	Doctoral Training Partnership	London Arts and Humanties Partnership	Prof. Jonathan Woolf	AH/L503873/1	159110	2014-01-10	2020-03-31	11074053.98
BBSRC	Doctoral Training Partnership	LIDO 2	Prof. Gabriel Waksman	BB/M009513/1	167165	2015-10-01	2023-09-30	20920514
BBSRC	Doctoral Training Grant - Medicine (Prof Rachel Chambers)	Macrophage Plasticity and Tissue Repair	Prof. Rachel Chambers	BB/P504440/1	172088	2016-10-01	2020-09-30	104696
BBSRC	Doctoral Training Grant - Dr. Rebecca	Cell Reproduction	Prof. Rebecca Shipley	BB/P504658/1	172166	2016-10-01	2020-09-30	104696

Detailed View

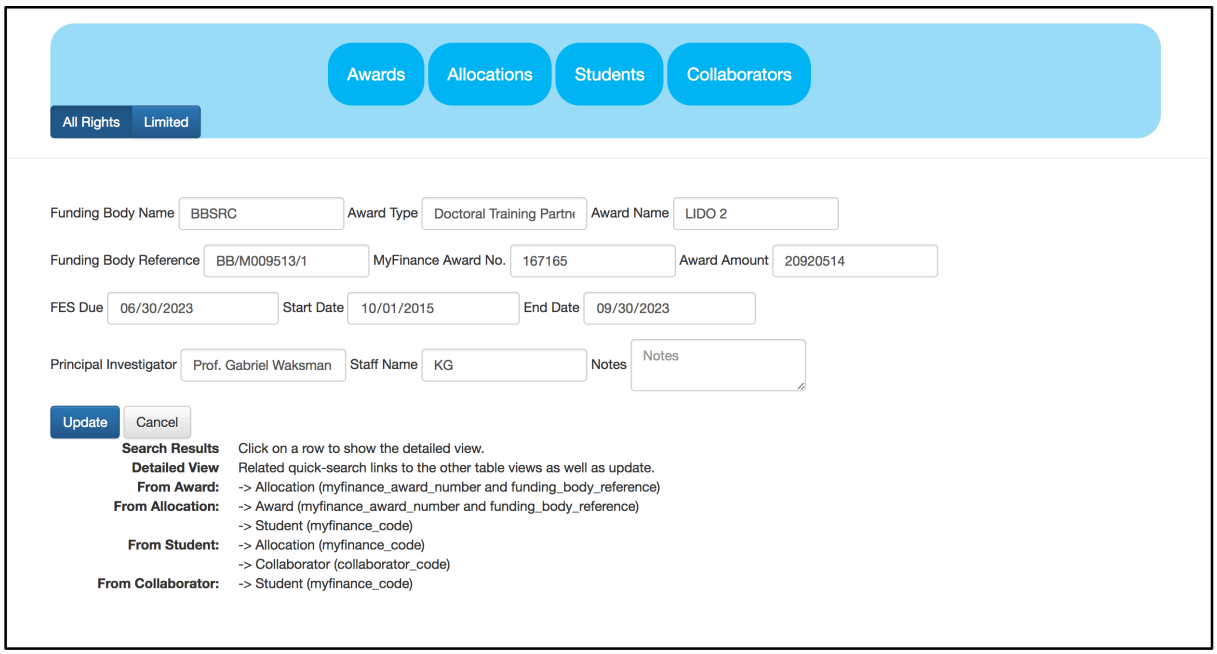
Awards | Allocations | Students | Collaborators

All Rights | Limited

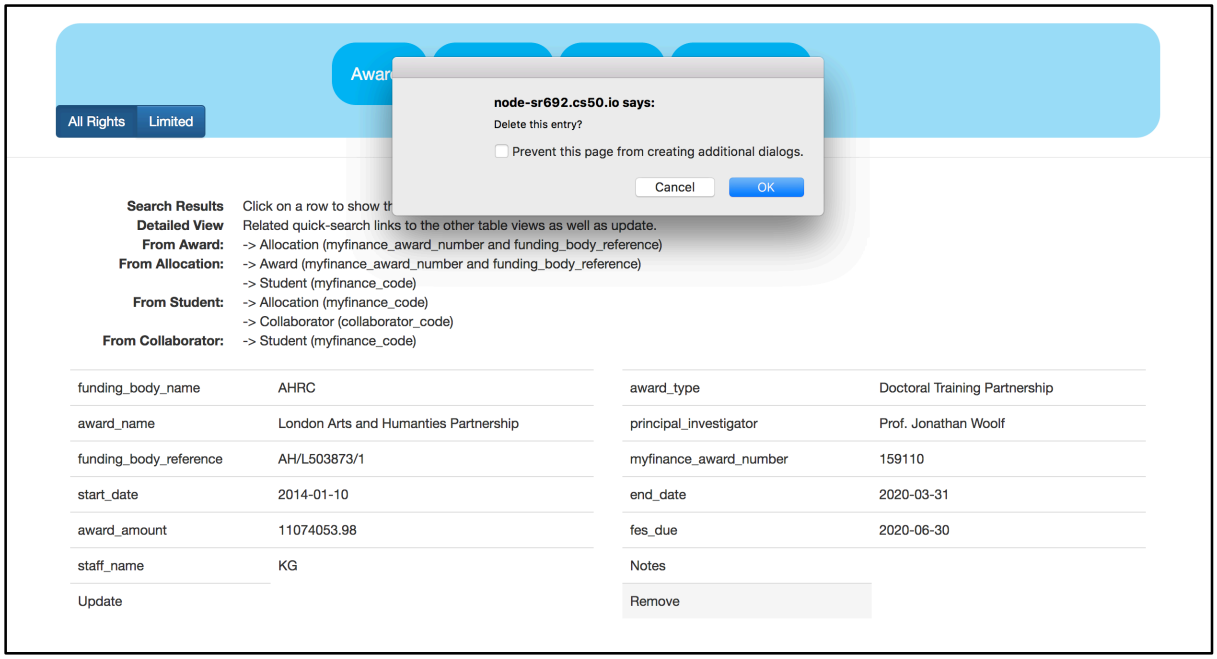
Search Results Click on a row to show the detailed view.
Detailed View Related quick-search links to the other table views as well as update.
From Award: -> Allocation (myfinance_award_number and funding_body_reference)
From Allocation: -> Award (myfinance_award_number and funding_body_reference)
-> Student (myfinance_code)
From Student: -> Allocation (myfinance_code)
-> Collaborator (collaborator_code)
From Collaborator: -> Student (myfinance_code)

funding_body_name	BBSRC	award_type	Doctoral Training Partnership
award_name	LIDO 2	principal_investigator	Prof. Gabriel Waksman
funding_body_reference	BB/M009513/1	myfinance_award_number	167165
start_date	2015-10-01	end_date	2023-09-30
award_amount	20920514	fes_due	2023-06-30
staff_name	KG	Notes	
Update		Remove	

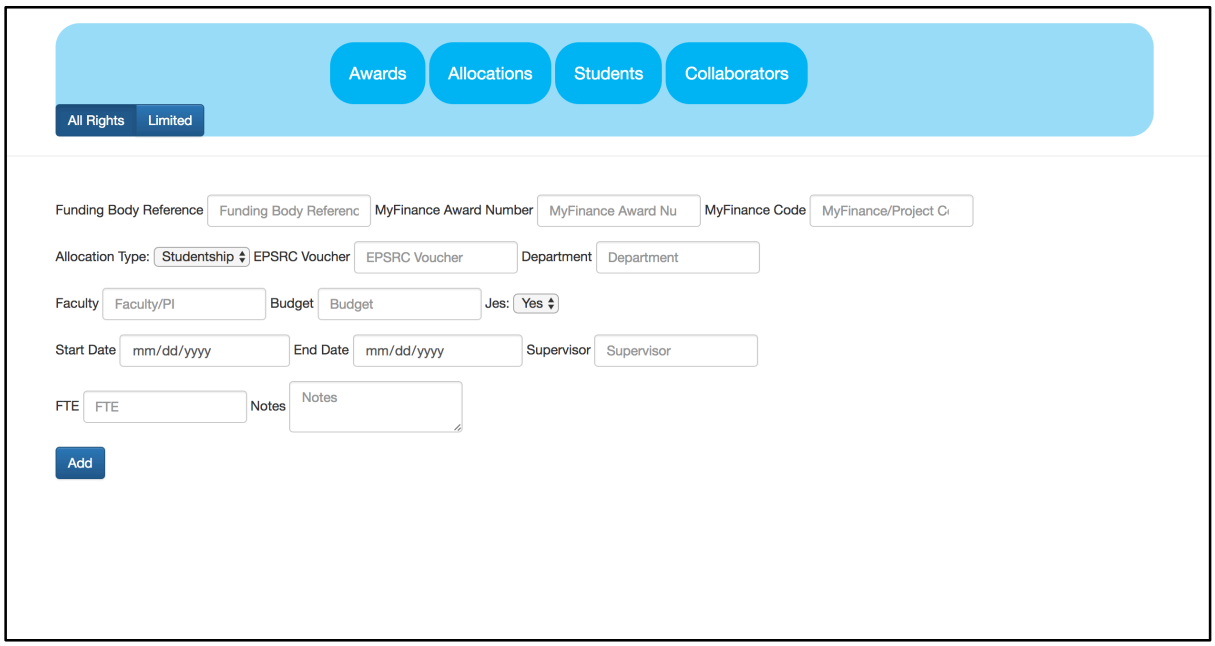
Updating a Record



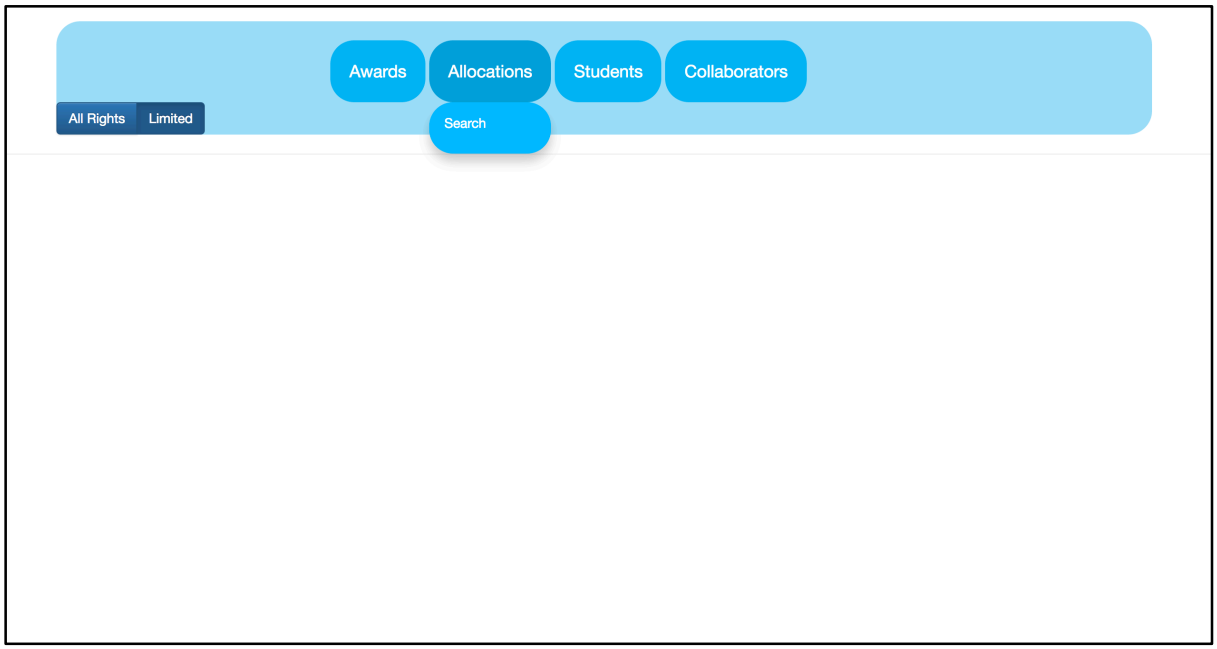
Removing a Record



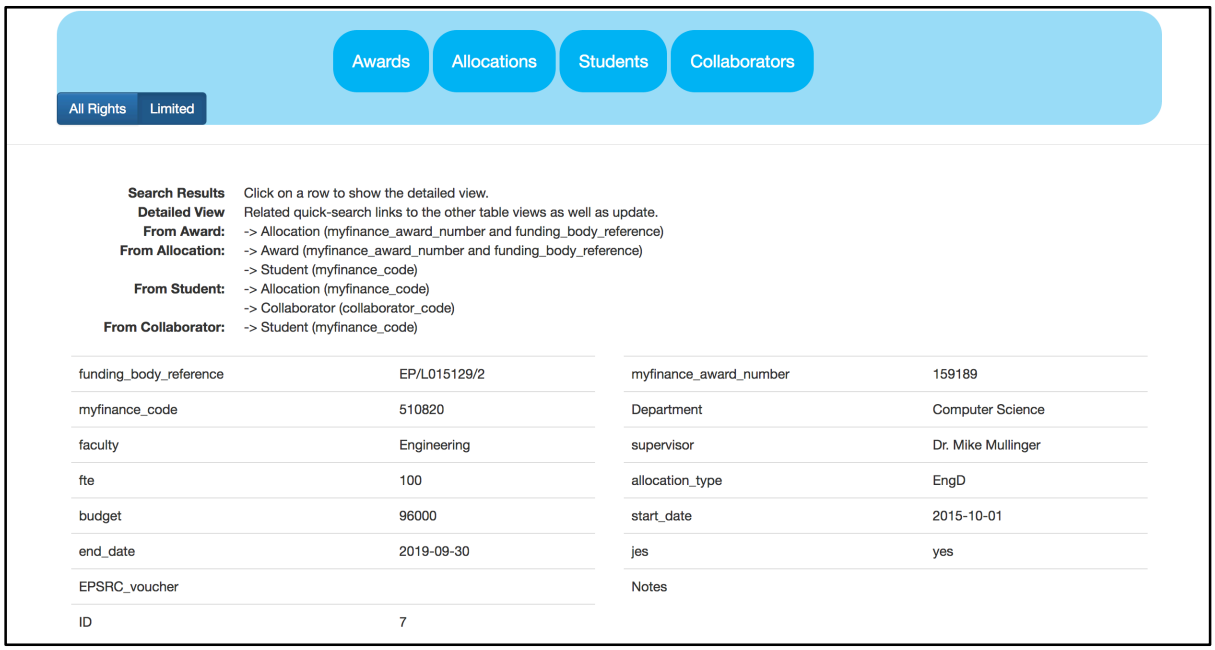
Adding a Record



Secondary User Home Page without Add



Secondary User Detailed View without Update/Remove



G. Bi-Weekly Reports

Bi-Weekly Report #1 - Research Grant and Fund Management System

Team 49: Allen Wang, Stylianos Rousoglou
Client: Mark Burgess
Date: January 30th, 2017

Overview

We met with Mark Burgess, our client, on Monday January 30th. We were introduced to the project idea which relates to Mark's work in Research Services here at UCL. Through a productive conversation, we made progress outlining some requirements and the reasons behind this project idea. We discussed high level design and conceptual outline for how to best solve Mark's problem.

Meetings

January 30, 2017

Mark first introduced us to the problem at hand and outlined the current system used to keep track of fund allocations, highlighting the need for a more sophisticated system going forward. Research grant organization and fund allocation currently takes up hundreds of excel spreadsheets, a method that the department has used for the past 20 years. As the funding increases and the number of PhD students now hovers around 5000, a more efficient information management system is badly needed; it would increase productivity for Mark's department as well as allow for data transparency between multiple departments.

Progress

We gained a high level idea of the problem at hand. Requirements were mapped out and split into two categories, "must" and "should" requirements, based on the client's immediate needs and other desires. We began thinking of technologies and possible application design approaches. Relevant background information regarding scope, use cases, and purpose were fleshed out via a productive initial meeting. Looked into node.js libraries for potential database (NoSQL, MySQL) integration.

Issues

Our understanding of the data is currently still lacking. We need to contact the client and discuss the significance of all the information available, as well as possible ways to condense it. In designing our database and implementing requirements, we need to completely understand how the data relates to each other and to the grand scheme of the project. We are currently unsure of details regarding implementation of back-end, deployment/hosting, and future maintenance.

Next two weeks

- Delegate tasks
- Establish deliverables and timelines
- Research and agree upon technology stack
- Narrow down our implementation from a high level to a more concrete idea
- Plan to also draft together an ERD or database design for the information
- Schedule a second meeting with Mark to clarify questions
- Initiate software development - test candidate libraries' usability, integration etc.

Individual Work:

Stelios

I focused on researching possible tools and technologies we may use in the product's lifecycle. Since we are leaning toward *Node.js* as our primary back-end development language, I read through the documentation of various MySQL APIs for Node.js on github, ultimately settling on the best-documented one. For a better grasp of the big picture, I also researched various deployment options. Specifically, I looked into Docker and continuous integration with Codeship. I also read more about Heroku's easy deployment package.

Allen

I arranged our meeting with Mark Burgess and gathered background information. With this information and Mark's consultation I wrote up some of the most necessary requirements. I plan to design the database system and clarify how the data should be viewed in our next meeting. I also looked at possible database managements systems (MySQL, NoSQL, GraphQL) and we are leaning towards MySQL.

Bi-Weekly Report #2 - Research Grant and Fund Management System

Team 49: Allen Wang, Stylianos Rousoglou
Client: Mark Burgess
Date: February 10th, 2017

Overview

The past two weeks were our first chance to really sit with the project and work out details, distill tool and design decisions, and set priorities. We met with Mr. Burgess on Tuesday, February 7th, and offered him the bulk of our questions/clarifications. Mr. Burgess went into detail about the specifics of his data and how to best manage it, the needs and wants of a working solution, and his vision of simple software that will nevertheless greatly facilitate his work, immediately. On our end, we wrote our first back-end and database code, and have been using test data provided by the client to model user cases.

Meetings

February 7, 2017

We met with Mr. Burgess for a whole hour to discuss in more detail the aspects of his job that are relevant to the development of the software solution. Having prepared a long list of detailed questions in advance, we spent the bulk of our time noting and highlighting Mark's explanations and clarifications. The client then took the time to describe the basic features and characteristics of a desired solution, going into extensive detail about querying and sample user cases. We believe our meeting was extremely productive, as we both left having a much more concrete idea of the problem at hand and the approaches to a solution.

Progress

In the past two weeks, we have:

- Decided the tools to be used in developing a solution
- Met with the client to establish requirements and priorities
- Explored and settled on Node.js libraries that will facilitate development
- Designed and created preliminary database files

- Integrated and tested database-server communication
- Written preliminary code for the product
- Arranged our next meeting with Mark Burgess and UCL's Information Systems Division (regarding deployment via UCL servers)

We think the project is running on time.

Issues

We are looking way into the future here, but given that the client's data is not only sensitive but also confidential, we have been proactive in discussing ways of hosting the software solution after it's been developed. Our client agreed that the best approach would be contacting the Information Systems Division of UCL and scheduling a meeting to discuss potential cooperation in deploying and maintaining the server.

Next two weeks

- Meet with the client in their office to see his work and more data
- Meet with the UCL Information Systems Division to discuss hosting
- Continue developing back-end
- Start discussing front-end and (minimalist) user interface
- Import some of client's data into code for testing

Individual work

Stelios

I focused on back-end development in Node.js and data storage with SQL. I wrote preliminary code to integrate the SQL server with the application's server. I created preliminary SQL tables to hold the client's data and enforce the appropriate rules on his dataset. I also wrote some back-end methods for manipulating the tester data of the client, which include asynchronous queries to and responses from the SQL server, and which are designed to be highly modular (for good style and future reusability.)

Allen

I outlined the MoSCoW Requirements and began thinking of database designs and how to store the information from excel into sql. I also set up a meeting with our client and the UCL IT team to make sure we work out the details for hosting and maintenance. The database design currently doesn't seem to be too complex since it's more about fetching and querying data that will be of use to our client.

MoSCoW Requirements

ID	Description	Priority
1	The new system is to manage the information from the current system's hundreds of spreadsheets, as well as updating adding information and notes	Must
2	Users have different privileges, managed by the client's department to allow read or read/write access	Must
3	Users can conduct different searches using different parameters to obtain a resultset for student, project, and grant information.	Must
4	Client department users can change a notes section for each grant, project, and student row, providing a form of version control and logging/consistency.	Must
5	Completed grants should be still recorded and searchable, per UCL guidelines in retaining records	Should
6	The system supports integration with portico/UCL since UCL owns the information - data integrity	Could
7	The system currently supports all of the RCUK grants, and eventually collaborators from external parties can also be tracked in this system	Won't

Tables and Keys

Name: Award

Description: A grant by the Research Council that is to be partitioned into projects and managed by UCL

Keys:

Funding Body Ref (Varchar) - Primary Key
MyFinance Award No. (Int) - Unique
Start Date - (Timestamp)
End Date - (Timestamp)
Amount - (Double)
Status - (Varchar)

Fes Due - (Varchar)

Name: Project

Description: Each grant maps to one or many projects that the grant money can be allocated for

Keys:

MyFinance Code (Int) - Unique
Type - (Varchar)
EPSRC Voucher (Varchar)
Award (Double)
Jes (Varchar)
Industry Account (Int)
Supervisor (Varchar)
Industrial partner - (Varchar)

Bi-Weekly Report #3 - Research Grant and Fund Management System

Team 49: Allen Wang, Stylianos Rousoglou
Client: Mark Burgess
Date: February 24th, 2017

Overview

The past two weeks we were able to plan and get a clearer outline of our preliminary research. We met with the UCL IT services to discuss hosting and future end goal implementations as well as developed our initial plans for the application implementation.

Meetings

February 21st, 2017

We met with Mr. Burgess and the UCL IT department consultants. They told us about the available database, front and backend technologies that we could consider since our client ultimately wants to make this available to UCL personnel and possibly integrated with Portico. They advised us about our proof of concept tools, including Java(Spring), PHP, Oracle, MySQL just to name a few. Ultimately we were able to relay our client's vision and requirements to them for a clear recommendation that this working prototype (proof of concept) would be best implemented using the technologies we deem the most fit for implementation. Along with hosting on cloud9, AWS, or Azure, we concluded that node.js and MySQL with JS framework/Bootstrap on the front-end would be best. This helps us accomplish our client's goal, which is to build a model and working prototype that can enable his department and the research council to assist in developing into a business application.

Progress

In the past two weeks, we have:

- Re defined the tools needed in our implementation
- Met with the client to clarify more requirements and understand the problem and solution
- Explored and settled on Node.js libraries that will facilitate development

- Reviewed and learned material on Angular/Bootstrap and other technologies needed for the project
- Sketched out UI that satisfies requirements and clarified our high-level vision
- Written preliminary code for the product
- Arranged our next meeting with Mark Burgess where more of the data will be showed to us in a demonstration to give us an experience with the old system UX and how we can improve it

We think the project is running on time.

Issues

Allen is not familiar with Front end development and has to quickly get comfortable with Angular and node + MySQL. Also there could be a bottleneck in development as most of the backend code depends on the front end's implementation.

Next two weeks

- Meet with the client in their office to see his work and more data
- Continue developing back-end
- Continue developing front-end and (minimalist) user interface
- Work on Data integrity before migration to our database

Individual work

Stelios

I continued to work on the initial code base, adding high-level back-end functionality and many generic endpoints to which requests will be made by the front-end in the future. I also revised the initial design of our database tables after a more comprehensive meeting with our client, and continued to give thought to how to best design various functionality for features to-be-developed.

Allen

I looked into the front end technologies and started learning via tutorial. I also sketched out a lot of the UI and looked at UI minimalist design examples that would accomplish our goal. During our meeting, we were able to satisfy our client's use case requirements with our UI design of searches and data querying.

Bi-Weekly Report #4 - Research Grant and Fund Management System

Team 49: Allen Wang, Stylianos Rousoglou
Client: Mark Burgess
Date: March 10th, 2017

Overview

In the past two weeks, we have started building out the front end and developing the must-have requirements. Currently we are able to process search queries and add entries to our database and have the UI implemented. We will continue to look to improve this aspect as well as finish up our server-side end points so this capability can be extended to all of the categories of our application. We also finally received excel-sheet formatted data to sample with our prototype demonstration. It had been an issue because a lack of data integrity prevented our client from giving us these sheets earlier.

Meetings

March 8th, 2017

We met with Mr. Burgess in his office, where we were shown a demonstration of his current information storage system. We also were shown different data sheets to give us more details to add into our application. Mr. Burgess also gave us a positive response to our developed UI which allowed us to move onto other parts of our application. He also suggested a few changes that we made sure to prioritize moving forward.

Progress

In the past two weeks, we have:

- Outlined our technical challenges and tasks to complete in time
- Developed solutions and began implementing them while dealing with debugging
- Code progressed on both ends of our system
- Arranged our next meeting with Mark Burgess where we'll show him the full demonstration we show in our lab

We think the project is running on time.

Issues

We have only just received excel sheets for our demo, and have less than a week to migrate them, as well as finishing up all of the javascript client and server side code to complete all the requirements.

Next two weeks

- Demonstrate progress and some functionality in lab and to our client
- Finish developing back-end
- Finish developing front-end

Individual work

Stelios

I modified our database structure and added a table after seeing a complete view of Mr. Burgess's information and data format. I also developed the backend and added more endpoints and implemented some functions of our requirements. I also began looking at how to migrate our client's new datasheets into our database.

Allen

I arranged the meeting with our and worked to incorporate his feedback into our UI design. After our meeting, I built up the UI and client-side javascript to start going through the requirements of our project. I made enough progress to where my strategies of satisfying the requirements have proven to be successful, and can move on implementing it for the various tables and data queries.

Bi-Weekly Report #5 - Research Grant and Fund Management System

Team 49: Allen Wang, Stylianos Rousoglou
Client: Mark Burgess
Date: March 23rd, 2017

Overview

In the past two weeks, we have nearly completed all of the must-have requirements and functionality of our project. We are capable of searching, adding, and updating entries to our database system and have been able to migrate normalized data from our client's excel sheets to our system. In the coming few weeks, we will clean up the User Interface as well as test our system with different inputs and user actions to try and find bugs that break our system. We will also start on our final submission deliverables.

Meetings

None Available

Mr. Burgess unfortunately was out of from the date after our in-lab demo on March 16th. We were unable to meet with him and show him our demo and working functionalities but with the approval of both our TA and Dr. Yun Fu, we believe that our progress will prove more than satisfactory for when Mr. Burgess returns.

Here is the automated response from Mr. Burgess's email.

Burgess, Mark
Wednesday, March 22, 2017 at 10:49 AM
To: Wang, Allen

I am now out of the office until 31 March 2017 inclusive. I will not respond to any emails received during these dates.

Please contact the following staff in my absence:

Progress

In the past two weeks, we have:

- Fulfilled almost all of our requirements
- Migrated all test data from Mr. Burgess's system into our database
- Iterated over our code to fix bugs and improve functionality
- Cleaned up and commented working code
- Presented a live demo with successful functionality to our TA and module staff
- Worked on code and library organization as well as user experience improvement

We think the project is running on time.

Issues

On the client side, our update function has to be redone to fix a bug. Also our UI is not as visually appealing as we would like. Color coding as well as other minor fixes to our front-end will likely yield a tremendous improvement in the User Experience. This was the main area of critique/feedback from our staff.

Next two weeks

- Improve the UI
- More testing
- Develop thorough documentation
- Work on all project deliverables

Individual work

Stelios

I worked on consolidating the back-end functionality and cleaning much of the code that had been there for several stages of development. I factored code out into Node.js package files for better organization and improved readability of the server-side program. I consolidated several methods to minimize repetition of SQL syntax and queries, as well as added comprehensive comments before all functions.

Allen

I made improvements to the front-end client javascript code and allowed us to satisfy almost all of our requirements. I also implemented additional features like quick-search

hyperlinks between results that make for a more pleasant UX. I began working on revamping the UI as well, and will also seek to organize and comment out the client-side code better after we've achieved all of our functionalities.