2. Project Description

The following document describes the fundamental scope and plan of a group project to be done in the Software engineering course, Software Process (331). The purpose of this project is to create an interactive multi-year schedule builder for software engineering students. In addition, being a team project, it emphasizes on the value of team cooperation and organization. The team name is “the Schedulers” and is composed of 12 students, having specific roles of front-end programmer, back-end programmer and documenters.

The document provides a foundation to follow when implementing the system. It contains what the system should do, meaning requirements and behaviors, and what we will use to achieve our goal, defining the resources and technologies used.

The overall goal of this project is to create a system that can generate a schedule. The application should allow a user to log in to the network after signing up. It shall also invite the user to add his/her classes and display his/her preferences, for example: choosing only night classes. The system shall create a schedule, based on the inputs of the user, to be displayed on screen.

3. GOALS AND CONSTRAINTS

3.1 FUNCTIONAL REQUIREMENTS

**1. Landing Page**

**2. Login**

**3. Registration**

**4. Account Management**

4.1 The system shall allow the user to view all account information within 2 seconds.

4.2 The user may modify mutable account information.

4.3. The system may generate a confirmation message describing account information changes.

4.4 The system shall update the user’s database with the account information changes.

**5. Record Management**

5.1 Upon first time login, the system shall allow the user the creation of a student record, based on the set of courses previously taken.

5.2 The user shall be able to update and modify the information provided initially as well as any information entered subsequently.

5.3 The system shall store the information entered during the creation or the modification of the record once the student commits to saving.

5.4 The system shall discard all entered information if the user's session is terminated or interrupted prior to committing to save.

5.5 The system shall allow the user to view the saved student record at any time during an active session.

5.6 The user may delete the existing student record at anytime during an active session, and may recreate a new one.

5.7 The system shall allow the existence of one and only one student record per user account.

**6. Course Selection**

6.1 The system shall allow registered and unregistered users alike to view the list of all courses and their respective descriptions and schedules.

**7. Schedule Preferences**

**8. Schedule Generation**

**9. Logout**

**10. Administrator**

10.1The administrator shall be able to add one or more courses to all programs.

10.2The administrator shall be able to edit all courses.

10.3The administrator shall be able to remove one or more courses from all programs.

10.4The administrator shall be able to add one or more sections to all courses.

10.5 The administrator shall be able to edit all sections.

10.6 The administrator shall be able to delete one or more sections from all courses.

10.7 The system shall save any and all modifications performed by the administrator, only if the commit to changes operation terminates successfully.

10.8 The system shall discard all changes if the administrator's session is terminated unexpectedly.

|  |  |  |  |
| --- | --- | --- | --- |
| Name: | Edit Account Information | | |
| Identifier: | UC | Version: | 1.0 |
| Date Created: | Feb 4, 2016 | Last Modified: |  |
| Importance: | 5 | | |
| Actor(s): | Registered User / Administrator | | |
| Goal: | To edit an existing account information. | | |
| Summary: | The user adds or modifies entries into the existing account information. The system stores the newly entered information and updates account. | | |
| Related use-cases: |  | | |
| Preconditions | 1. User is logged on.  3. System has accessed the account menu. | | |
| Trigger: | User activates the “ Edit Account Information” process. | | |
| Basic Flow: | 1. User selects the “Edit Account Information” Option.  2. System prompts to enter information.  3. User inputs information.  4. User commits to saving.  5. System responds with outcome of the operation. | | |
| Post-Conditions: | **Success:** Student record is Updated successfully.  **Failure:** System fails to process task and displays an error message. | | |
| Minimum Guarantee: | System remains unchanged. | | |
| Risk Assessment: | Low | | |
| Notes: |  | | |
| Author(s): | Adil Hssaini | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Name: | Create Record | | |
| Identifier: | UC | Version: | 1.0 |
| Date Created: | Feb 4, 2016 | Last Modified: |  |
| Importance: | 5 | | |
| Actor(s): | Registered User / Administrator | | |
| Goal: | To create a student record to be stored by the system | | |
| Summary: | The user provides the system with a set of previously taken courses. The system stores that information and uses it to create a record for the specified account. | | |
| Related use-cases: | UC **“Edit Account Information”** | | |
| Preconditions | 1. User is logged on.  2. System has accessed the account menu.  2. System has no existing record associated with the account. | | |
| Trigger: | User activates the “Create Record” process | | |
| Basic Flow: | 1. User selects the “Create Record” Option  2. System prompts to enter information.  3. User inputs information.  4. User commits to saving.  5. System responds with outcome of the operation. | | |
| Post-Conditions: | **Succcess:** Student record is created successfully.  **Failure:** System fails to process task and displays an error message. | | |
| Minimum Guarantee: | System remains unchanged. | | |
| Risk Assessment: | Low | | |
| Notes: |  | | |
| Author(s): | Adil Hssaini | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Name: | Update Record | | |
| Identifier: | UC | Version: | 1.0 |
| Date Created: | Feb 4, 2016 | Last Modified: |  |
| Importance: | 4 | | |
| Actor(s): | Registered User / Administrator | | |
| Goal: | To update an existing student record. | | |
| Summary: | The user adds or modifies entries into the student record. The system stores the newly entered information and updates record. | | |
| Related use-cases: |  | | |
| Preconditions | 1. User is logged on.  3. System has the record menu. | | |
| Trigger: | User activates the “ Update Record” process | | |
| Basic Flow: | 1. User selects the “Update Record” Option  2. System prompts to enter information.  3. User inputs information.  4. User commits to saving.  5. System responds with outcome of the operation. | | |
| Post-Conditions: | **Succcess:** Student record is updated successfully.  **Failure:** System fails to process task and displays an error message. | | |
| Minimum Guarantee: | System remains unchanged. | | |
| Risk Assessment: | Low | | |
| Notes: |  | | |
| Author(s): | Adil Hssaini | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Name: | Delete Record | | |
| Identifier: | UC | Version: | 1.0 |
| Date Created: | Feb 4, 2016 | Last Modified: |  |
| Importance: | 2 | | |
| Actor(s): | Registered User / Administrator | | |
| Goal: | To delete an existing student record. | | |
| Summary: | The user deletes the existing student record associated with account. The system has no record associated with the account afterwards. | | |
| Related use-cases: | UC **“Update Record”** | | |
| Preconditions | 1. User is logged on.  3. System has accessed the record menu. | | |
| Trigger: | User activates the “ Delete Record” process | | |
| Basic Flow: | 1. User selects the “Delete Record” Option  2. System prompts for confirmation.  3. User commits to deleting.  4. System responds with outcome of the operation. | | |
| Post-Conditions: | **Succcess:** Student record is deleted successfully.  **Failure:** System fails to process task and displays an error message. | | |
| Minimum Guarantee: | System remains unchanged. | | |
| Risk Assessment: | Low | | |
| Notes: |  | | |
| Author(s): | Adil Hssaini | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Name: | View Courses | | |
| Identifier: | UC | Version: | 1.0 |
| Date Created: | Feb 4, 2016 | Last Modified: |  |
| Importance: | 3 | | |
| Actor(s): | All users | | |
| Goal: | To view information about a course | | |
| Summary: | The user can browse through the list of offered courses and is able to view their corresponding descriptions, and schedules. | | |
| Related use-cases: |  | | |
| Preconditions | 1. User is on the landing page. | | |
| Trigger: | User activates the “ View Courses” process | | |
| Basic Flow: | 1. User selects the “Browse List” Option.  2. System displays page with list of programs.  3. User selects program.  4. System displays list of courses required for the program.  5. User selects course.  4. System responds with outcome of the operation. | | |
| Post-Conditions: | **Succcess:** Course information is displayed successfully.  **Failure:** System fails to process task and displays an error message. | | |
| Minimum Guarantee: | System remains unchanged. | | |
| Risk Assessment: | Low | | |
| Notes: |  | | |
| Author(s): | Adil Hssaini | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Name: | Add Course to Program | | |
| Identifier: | UC | Version: | 1.2 |
| Date Created: | Feb 2, 2016 | Last Modified: |  |
| Importance: | 5 | | |
| Actor(s): | Administrator | | |
| Goal: | To add a new course to a specific program. | | |
| Summary: | The Administrator updates the list of required courses for a specific program by adding a new course. | | |
| Related use-cases: |  | | |
| Preconditions | 1. Actor is logged on as administrator.  2. System has accessed the program menu.  3. The course is not part of the program sequence stored by the system. | | |
| Trigger: | Administrator activates the “Add Course to Program” process. | | |
| Basic Flow: | 1. Administrator selects the “Add Course to Program” option.  2. System prompts for information about course.  3. Administrator inputs information.  4. Administrator commits to adding the course.  5. System responds with the outcome of the operation. | | |
| Post-Conditions: | **Success:** Course is added successfully to the program listing.  **Failure:** The system fails to process the task and displays an error message. | | |
| Minimum Guarantee: | List of courses in the program stored by the system will not be affected. | | |
| Risk Assessment: | Low | | |
| Notes: |  | | |
| Author(s): | Adil Hssaini | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Name: | Edit Course | | |
| Identifier: | UC | Version: | 1.2 |
| Date Created: | Feb 2, 2016 | Last Modified: |  |
| Importance: | 5 | | |
| Actor(s): | Administrator | | |
| Goal: | To edit a course's description or information. | | |
| Summary: | The administrator modifies the existing description or details of a course. | | |
| Related use-cases: |  | | |
| Preconditions | 1. Actor is logged on as administrator.  2. System has accessed the course menu. | | |
| Trigger: | Administrator activates the “Edit Course” process. | | |
| Basic Flow: | 1. Administrator selects the “Edit Course” option.  2. System prompts for new information.  3. Administrator inputs information.  4. Administrator commits to Editing the course.  5. System responds with the outcome of the operation. | | |
| Post-Conditions: | **Success:** Course is edited successfully.  **Failure:** The system fails to process the task and displays an error message. | | |
| Minimum Guarantee: | All other courses present in the list of courses required for the program will not be affected. | | |
| Risk Assessment: | Low | | |
| Notes: |  | | |
| Author(s): | Adil Hssaini | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Name: | Delete Course | | |
| Identifier: | UC | Version: | 1.2 |
| Date Created: | Feb 2, 2016 | Last Modified: |  |
| Importance: | 5 | | |
| Actor(s): | Administrator | | |
| Goal: | To remove a course from a program listing. | | |
| Summary: | The Administrator deletes a course from the list of required courses for a specific program. | | |
| Related use-cases: |  | | |
| Preconditions | 1. Actor is logged on as administrator.  2. System has accessed the course menu. | | |
| Trigger: | Administrator activates the “Delete Course” process. | | |
| Basic Flow: | 1. Administrator selects the “Delete Course” option.  2. System prompts for confirmation.  3. Administrator commits to deleting the course.  4. System responds with the outcome of the operation. | | |
| Post-Conditions: | **Success:** Course is deleted successfully.  **Failure:** The system fails to process the task and displays an error message. | | |
| Minimum Guarantee: | All other courses present in the list of courses required for the program will not be affected. | | |
| Risk Assessment: | Low | | |
| Notes: |  | | |
| Author(s): | Adil Hssaini | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Name: | Add Section | | |
| Identifier: | UC | Version: | 1.2 |
| Date Created: | Feb 2, 2016 | Last Modified: |  |
| Importance: | 5 | | |
| Actor(s): | Administrator | | |
| Goal: | To add a course section. | | |
| Summary: | The administrator adds a section to a course in the list of required courses for a program. | | |
| Related use-cases: | UC – **Edit Course** | | |
| Preconditions | 1. Actor is logged on as administrator.  2. System has accessed the course menu. | | |
| Trigger: | Administrator activates the “Add Section” process. | | |
| Basic Flow: | 1. Administrator selects the “Add Section” option.  2. System prompts for section details.  3. Administrator inputs information.  4. Administrator commits to adding the section.  5. System responds with the outcome of the operation | | |
| Post-Conditions: | **Success:** Section is created successfully.  **Failure:** The system fails to process the task and displays an error message. | | |
| Minimum Guarantee: | All courses and sections previously stored by the system will not be affected. | | |
| Risk Assessment: | Low | | |
| Notes: |  | | |
| Author(s): | Adil Hssaini | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Name: | Edit Section | | |
| Identifier: | UC | Version: | 1.2 |
| Date Created: | Feb 2, 2016 | Last Modified: |  |
| Importance: | 5 | | |
| Actor(s): | Administrator | | |
| Goal: | To edit information or details of a section. | | |
| Summary: | The administrator modifies the existing details of a section. | | |
| Related use-cases: | UC – **Edit Course** | | |
| Preconditions | 1. Actor is logged on as administrator.  2. The system has accessed the section menu. | | |
| Trigger: | Administrator activates the “Edit Section” process. | | |
| Basic Flow: | 1. Administrator selects the “Edit Section” option.  2. System prompts for section's new details.  3. Administrator inputs information.  4. Administrator commits to editing the section.  5. System responds with the outcome of the operation | | |
| Post-Conditions: | **Success:** Section details are modified successfully.  **Failure:** The system fails to process the task and displays an error message. | | |
| Minimum Guarantee: | All courses and sections previously stored by the system will not be affected. | | |
| Risk Assessment: | Low | | |
| Notes: |  | | |
| Author(s): | Adil Hssaini | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Name: | Delete Section | | |
| Identifier: | UC | Version: | 1.2 |
| Date Created: | Feb 2, 2016 | Last Modified: |  |
| Importance: | 5 | | |
| Actor(s): | Administrator | | |
| Goal: | To delete a course section. | | |
| Summary: | The administrator deletes a course section from the list of available sections. | | |
| Related use-cases: | UC - “**Edit Course”** | | |
| Preconditions | 1. Actor is logged on as administrator.  2. System has accessed to the section menu. | | |
| Trigger: | Administrator activates the “Delete Section” process. | | |
| Basic Flow: | 1. Administrator selects the “Delete Section” option.  2. System prompts for confirmation.  3. Administrator commits to deleting the section.  4. System responds with the outcome of the operation | | |
| Post-Conditions: | **Success:** Section is deleted successfully.  **Failure:** The system fails to process the task and displays an error message. | | |
| Minimum Guarantee: | All courses and other sections previously stored by the system will not be affected. | | |
| Risk Assessment: | Low | | |
| Notes: |  | | |
| Author(s): | Adil Hssaini | | |



Figure 1: Administrator use case diagram

3.2 NON FUNCTIONAL REQUIREMENTS

**. Security**

The system shall be developed and coded with security at the forefront of concerns. The scheduler is implemented with React as a web application hosted by an Apache Web Server, and therefore provides default configurations as well as custom configurations that significantly help reduce XSS vulnerabilities and prevent attacks such as information leakage or PHP injections.

**2. Privacy**

To enforce and ensure privacy, practices such as session locks and expiry logouts shall be enforced. An encryption algorithm will also be implemented to provide an additional layer of abstraction to registered user's data.

**3. Maintainability**

The scheduler has several features that render its maintenance process more flexible than similar products. The use of React is in line with the nature of the system based on data that changes overtime. In addition, JavaScript being a very flexible and powerful language, makes expansions and additional options easily feasible as opposed to the standard directives or templates.

**4, Compatibility**

The system shall be compatible with all mainstream platforms browsers.

The system shall work on innate as well as on virtual machines.

The system shall be accessed world wide unless restricted by recipients' service providers or network settings.

**5. Portability**

The scheduler is not linked to any specific database. It is designed to operates as a public interface enabling access to multiple universities' programs. Options to link to selected databases are possible and are taken into account by the design.

**6. Performance**

The scheduler guarantees operations execution with minimal complexity. Coding aims at optimizing hardware and therefore reducing the system's and user devices' response time.

4. RESOURCE EVALUATION

4.1 HUMAN RESOURCES

|  |  |
| --- | --- |
| Team Member | **Le Vinh Dang** |
| Role | Back-End Programmer |
| Knowledge | SQL, C++, Ruby |
| Experience | * Teamwork experience * Object-oriented programming * Work experience as software developer |
| Strengths | * Debugging * Problem solving * SQL * Object Oriented |
| Availability | Any Time |

|  |  |
| --- | --- |
| Team Member | **Dias Marat** |
| Role | Back-End Programmer |
| Knowledge | Java, Javascript, HTML5, CSS3, PHP, Arduino, Objective C, C# |
| Experience | * Developed IPhone Applications and video games for several years * Developed server side applications using NodeJS and worked on backend previously * Team projects |
| Strengths | * Debugging * Problem solving * Mobile Development and Web Development |
| Availability | Anytime |

|  |  |
| --- | --- |
| Team Member | **Sean Marcoux** |
| Role | Front-end Lead |
| Knowledge | React, Javascript, PHP, HTML, CSS, Java |
| Experience | * 1 work term * Work experience developing React application in teams * Work experience developing improvements and bug fixes for a Java, eclipse RCP project |
| Strengths | * Developing refined user experiences * Debugging * Problem Solving * Java + React |
| Availability | Any time |

|  |  |
| --- | --- |
| Team Member | **Bruce Edouard Brazier** |
| Role | Back-end Lead |
| Knowledge | Javascript, PHP, HTML, CSS, Java, Bash,SQL |
| Experience | * 1 work term * Work/Personal Experience maintaining a website (Client/Server side) * Work experience working with Software Deployments and Scripting |
| Strengths | * Troubleshooting * Process Automation * Team coordination |
| Availability | Any time |

|  |  |
| --- | --- |
| Team Member | **Adriel Fabella** |
| Role | Front-end programmer |
| Knowledge | Javascript, PHP, SQL , HTML, CSS, Java. |
| Experience | * Part-time web designing and developing * Volunteer experience with PHP/ MySQL * Team Projects |
| Strengths | * Web developing * Communication skills * Problem solving |
| Availability | Any time |

|  |  |
| --- | --- |
| Team Member | **Adil Hssaini** |
| Role | Design & Development |
| Knowledge | C++, java, PHP, SQL, HTML, CSS |
| Experience | * Software maintenance experience * Prior involvement in large projects |
| Strengths | * Time and resource management * Versatile background * Team organization |
| Availability | 4 to 6 hours per week |

|  |  |
| --- | --- |
| Team Member | **Emili Vasseva** |
| Role | Team leader and documenter |
| Knowledge | Javascript, PHP, SQL , HTML, CSS, Java, C++, Prolog, Lisp, AspectJ |
| Experience | * Contractual web developer using HTML, CSS, JavaScript, PHP, MySQL * Currently developing a small game in Unity * Team Projects |
| Strength | * Web development * Problem solving * Leadership * Object-oriented programming |
| Availability | Any time |

4.2 TECHNICAL RESOURCES

The following section has been divided into several subsections which discuss technical resources associated with documentation, programming languages and software, hardware, operating systems, communication and management.

**4.2.1 Documentation**

The softwares that are being used for editing and reviewing the source documentations and codes are Google Docs/Drive and Microsoft word. Google Docs is a flexible program that allows teammembers to brainstorm and work together effectively on the same document in real-time. Google Drive is used to store Google docs files in a shared folder that all team members could access. Microsoft Word was used for more individual documentation from team members and for greater flexibility in organizing documented information. Adobe Reader was utilized in order to render documents into a format that could be read by all team members and their affiliates. WireFrame was used to design and showcase the front end interfaces and pages that would become part of the completed product.

**4.2.2 Programmin Languages and software**

The web server that is used to store the database is the WampServer64 2.5. The WAMP server, which is an accronym for Windows, Apache, MySQL and PHP, provides those programs for use. With the WampServer64 version 2.5, it will support the Apache : 2.4.9, the MySQL : 5.6.17, the PHP : 5.5.12, the PHPMyAdmin : 4.1.14, the SqlBuddy : 1.3.3, and the XDebug : 2.2.5. The programming languages which will be used to construct the final product are HTML, PHP, JavaScript and CSS.

**4.2.3 Hardware**

In addition to the above programs laptops and desktops will be used to install and carry the WAMP database and other software that will be required for the project. The laptops are not of a uniform variety but instead come from a wide range of manufacturers. They are DELL, HP and MAC laptops and desktops. In addition to containing the above serve the laptops and desktops also contain React, a JavaScript software used to develop the front end of the final product.

**4.2.4 Operating Systems**

In order to facilitate uniformity amongst team members the team laptops/desktops must have the following minimum requirements:

- 150 GB of memory

- 4 GB of RAM

- Intel Premium 4 or AMD Athlon x64

- WI-FI internet access

- Windows 7/Linux operating systems

- Headsets, earphones and speaker setups (to allow for discussions through skype)

**4.2.5 Communication and Management**

All code and document files related to the project are sorted and stored on Github under the repository Schedule-Builder through the use of Github accounts. All team members have access to Github folders, files and their content. To facilitate communication between teammembers easier, Facebook, Slack and Skype are used for holding discussions and meetings.

5. SCOPING

5.1 SCOPED IN

5.2 SCOPED OUT

6. SOLUTION SKETCH

6.1 ARCHITECTURE

Unlike the standard MVC architecture, the model, the view and the controller are not taken care by the same framework. In our architecture, the view is handled using React and data manipulation as well as database queries, are handled by Laravel. This means that the application is divided between client side and the server side. The client side handles everything to do with the view (everything the user sees and interacts with) while the server side handles everything else.

6(3)(1) (1).png

**Server Side**

For the server side, the components are the users( students, admin), the schedule, the database and the courses. The database contains the information on the students, the administrators and the courses. When the users modify their preferences and their information, it is updated in the database by a query. The students interact with the schedule component when they generate their schedule based on their preferences. The schedule component then fetches their preferences and generate the appropriate schedule. The administrator component can manage the courses and their properties. After the required information is gathered through the user page, the appropriate courses are then modified with a database query. Finally, the schedule provides the client side with the data that is to be displayed on the pages such as the student schedule or the full course sequence. This setup allows us to control the information of the students and the administrators. Ensuring that when they are needed, they can be accessed through the database. This will also facilitate the process of generating the algorithm since the components are independent and will be easier to manipulate.

**Client Side**

The main components for the client side are the UIManager and the components for each page. These will all be React components. The UIManager will be the necessary main React component and it will handle switching between all of the pages and hold the data that is common to all of them: the active user and if that user is an admin. This structure is the best way to handle the UI because switching between pages will be as simple as changing which component is being rendered. It also allows simple communication between pages through the UIManager.

The page components are the log-in, preference, account info, schedule, and admin pages. The log-in page needs to keep track of any input the user enters, which is username, password, and e-mail (if the user is registering for the first time). The account info page needs the same info, but this needs to be the information obtained from the server. The preferences page is where the user sets the courses they’ve taken, the courses they still need to take, and the preferences they have for their schedule. The preferences page component will keep track of all this info and will obtain any of it from the server if the user already input preferences in the past. From the preferences page, the user can click a button to build the schedule. This will generate the schedule on the server side and return the schedule information on the schedule page. The first semester classes here is a separate variable because these classes need the additional information of time, classroom, section, and teacher. The remaining semesters will simply be a list of classes for their recommended course sequence. Finally, the admin page will obtain a list of all courses and a list of all users registered in the database, so the admin can edit them.

6.2 TECHNOLOGIES IN USE

Programming Languages:

**HTML**

HTML is a computer and markup language that allows to create web sites and web documents. This language will consists of the very backbone of the website, when it comes to filling up the webpage with text and dialogs.

**CSC**

CSC is language within the markup language that allows to manipulate the design of the web document, meaning positioning, color and overall presentation.

**JavaScript**

JavaScript is the main client side programming language used for creating interactive websites. JavaScript support is built right into all the major browsers and can support object oriented programming. JavaScript will be the main dynamic language used by the front end team and everyone in the team possesses experience working with JavaScript.

**React**

React is an open source JavaScript library which contains a template language and some function hooks to efficiently render HTML. React manages all UI updates when data has been changed and will update only those changed data. This if efficient because the user can tell how a component will render by looking at one source file. A program flow does not need to be traced which can be efficient when working in a big team.

**PHP**

PHP is a server side scripting programming language used for web development. It is very well documented and can support objected oriented programming. PHP will be used for the backend and most of the team members have experience with PHP.

Framework:

**Laravel**

Laravel is an open source PHP web application framework that allows rapid development of web applications. Laravel uses MVC architecture and has features such as module package manager, template engine, database seeding, routes, authentication, and object oriented design. This will provide clean and manageable code.

Integrated Development Environments/Editors:

**PhpStorm**

PhpStorm is an Integrated Development Environment for Windows and Mac OS that allows developers to code their projects in PHP. It has syntax highlighting , plugins, different type of frameworks supported such as Symfony, Laravel, CakePHP, built in support for databases ,version control, debugging and testing. PhpStorm increases the productivity of developers and helps to get done things faster.

Source Code and Revision Management:

**Git**

Git is a source code management system used for software development. It allows developers to save different versions of their projects at different points of time and compare these versions between each other. Git allows developers to contribute to a repository(project) even if the developer is not connected to the Internet. It stores a local copy of the project on the local repository and changes made on the local repository can be pushed to the main repository. This allows to stay organized and maintain previous versions of the project.

Collaboration Software:

**GitHub**

Github is a website that hosts Git repositories and has all the functionalities of Git. It provides bug tracking, feature requests and wikis for projects. It is used in this project as it allows easier collaboration between developers.

**Google Docs**

Google Docs is an online word processor that allows individuals to edit and collaborate on documents in real-time. It is free and can be accessed by anybody.

**Draw.io**

Draw.io is a software application that allows to draw domain model, UML, use cases diagrams and etc. It allows collaboration between individuals and can be used as a plugin to Google Drive.

**MockFlow – Wireframe pro**

MockFlow – Wireframe pro is an application used to create the mockup of the website. It allows for collaboration where the whole team contributes to creating the backbone using the available widgets and elements and to display comments and reactions.

**Skype**

Skype is a free video chat application that allows users to do video conference calls and exchange documents. For this project it is used between sub sections of the teams because it is more efficient.

7. PLAN

Deliverable 2: March 9th

|  |  |
| --- | --- |
| Activity | Detailed Architecture |
| Purpose | Overall structure of the system |
| Artifact #1 | 4+1 Architectural View |
| Description | High-level structure of the system, composed of 5 views: logical view, process view, development view, physical view and scenarios. Used to describe a large system into multiple subsystems. |
| Work hour per person |  |
| Artifact #2 | Subsystems Interface Specifications/Module Interface Specifications |
| Description | Description of each subsystems meant to complete specific services, and their parameters (invalid/valid values) passed in functions. |
| Work hour per person |  |
| Start/End dates | February 11th - February 29th |
| Participants | Sean, Bruce, Emili, Nick |

|  |  |
| --- | --- |
| Activity | Detailed Design |
| Purpose | Complete class description of each subsystem |
| Artifact #1 | UML Class Diagram |
| Description | Connection between classes of each subsystems |
| Work hour per person |  |
| Artifact #2 | Dynamic Design Scenarios |
| Description | 2 dynamic design of 2 uses cases (using at least 3 system operations). This includes system sequence, operational contracts, and sequence diagrams. |
| Work hour per person |  |
| Artifact #3 | Estimation |
| Description | Estimated cost for integration, testing and documentation for each module. |
| Work hour per person |  |
| Due date | February 24th - March 8th |
| Participants | Adil, Alex, Salma, Ying-Chen |

|  |  |
| --- | --- |
| Activity | Rapid Prototyping |
| Purpose | Programming of the prototype designed using the architecture and design description. |
| Artifact #1 | Rapid Prototyping report |
| Description | Listing and commenting on classes/modules/drivers used for the rapid prototype. |
| Work hour per Deperson |  |
| Artifact #1 | Testing |
| Description | Testing code and report of the rapid prototype |
| Work hour per person |  |
| Artifact #3 | Risks |
| Description | Update of the risks, cost estimate and scoping from the deliverable 1 to deliverable 2 |
| Work hour per person |  |
| Due date | February 24th - March 8th |
| Participants | Dias, Adriel, Gabriele, Le Vinh |

Deliverable 3: April 6th

|  |  |
| --- | --- |
| Activity | Final prototype |
| Purpose | Final prototype of the a fully working software |
| Artifact #1 | Test report and instructions manuals |
| Description | 1. Test report on the entire making of the system  2. Instruction manual for future users |
| Work hour per person |  |
| Due date | March 10th - March 31st |
| Participants | Sean, Gabriele, Adriel, Le Vinh, Bruce, Dias |

|  |  |
| --- | --- |
| Activity | Testing Report |
| Purpose | Final report on all testing done on the final product |
| Artifact #1 | Test coverage |
| Description | 1. Listing of all tested items, and why.  2. Identification of 5 classes/methods and why they were tested. |
| Work hour per person |  |
| Artifact #2 | Test cases |
| Description | 1. Two mid-level units tests, with their respective test cases and descriptions.  2. Requirements testing and their test cases  3. Test cases of potential extreme system usages, and their respective description  4. Test testing regarding the security of the system |
| Work hour per person |  |
| Due date | March 10th - April 1st |
| Participants | Emili, Adil, Salma, Nick |

|  |  |
| --- | --- |
| Activity | System delivery |
| Purpose | Instructions on the system |
| Artifact #1 | Installation Manual |
| Description | Step by step instructions on how to install the system. |
| Work hour per person |  |
| Artifact #2 | Users Manual |
| Description | Step by step instructions on how to use the system. |
| Work hour per person |  |
| Due date | April 1st - April 5th |
| Participants | Alex, Ying-Chen |

|  |  |
| --- | --- |
| Activity | Final cost estimate |
| Purpose | Final coverage on the total amount of hours and money spent on the project |
| Artifact #1 | Working hours |
| Description | Final coverage on the number of hours put into the project by each person. |
| Work hour per person |  |
| Artifact #1 | Cost |
| Description | Final coverage on the costs spend on each individuals work and for technological resources |
| Work hour per person |  |
| Due date | April 1st - April 5th |
| Participants | Ying-Chen, Alex |

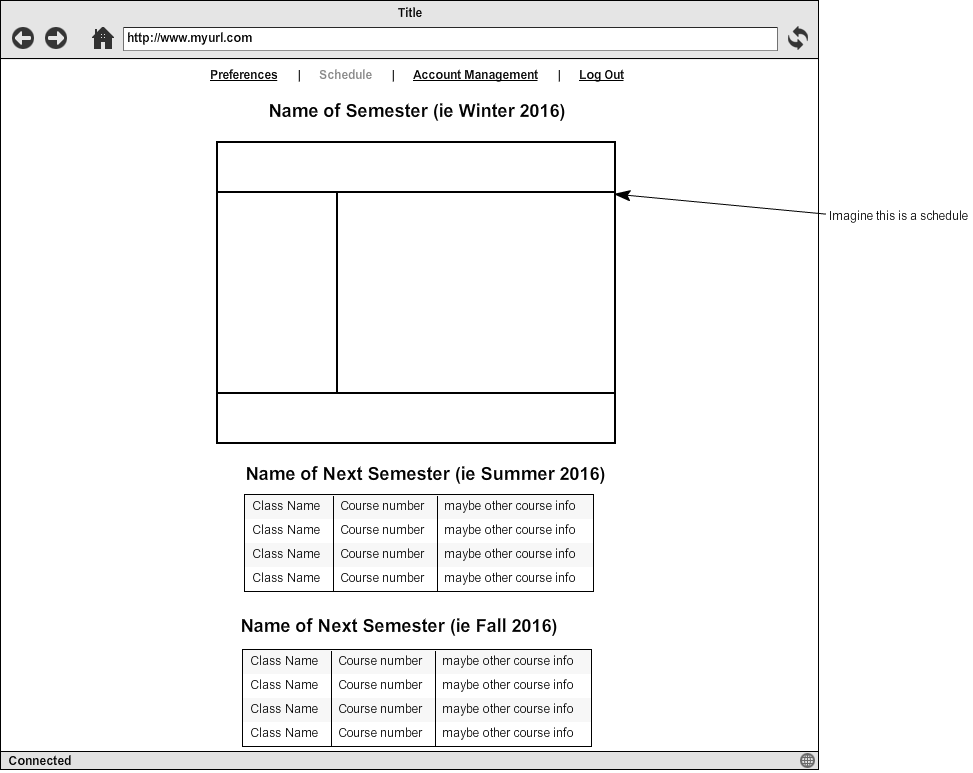
April 13th

|  |  |
| --- | --- |
| Activity | Finalization of the deliverable |
| Purpose | Completion of the project by submitting a complete and corrected report. |
| Artifact #1 | Final report |
| Description | Assembling and correction over the report and its content. |
| Work hour per person |  |
| Due date | April 7th - April 12th |
| Participants | Entire team |

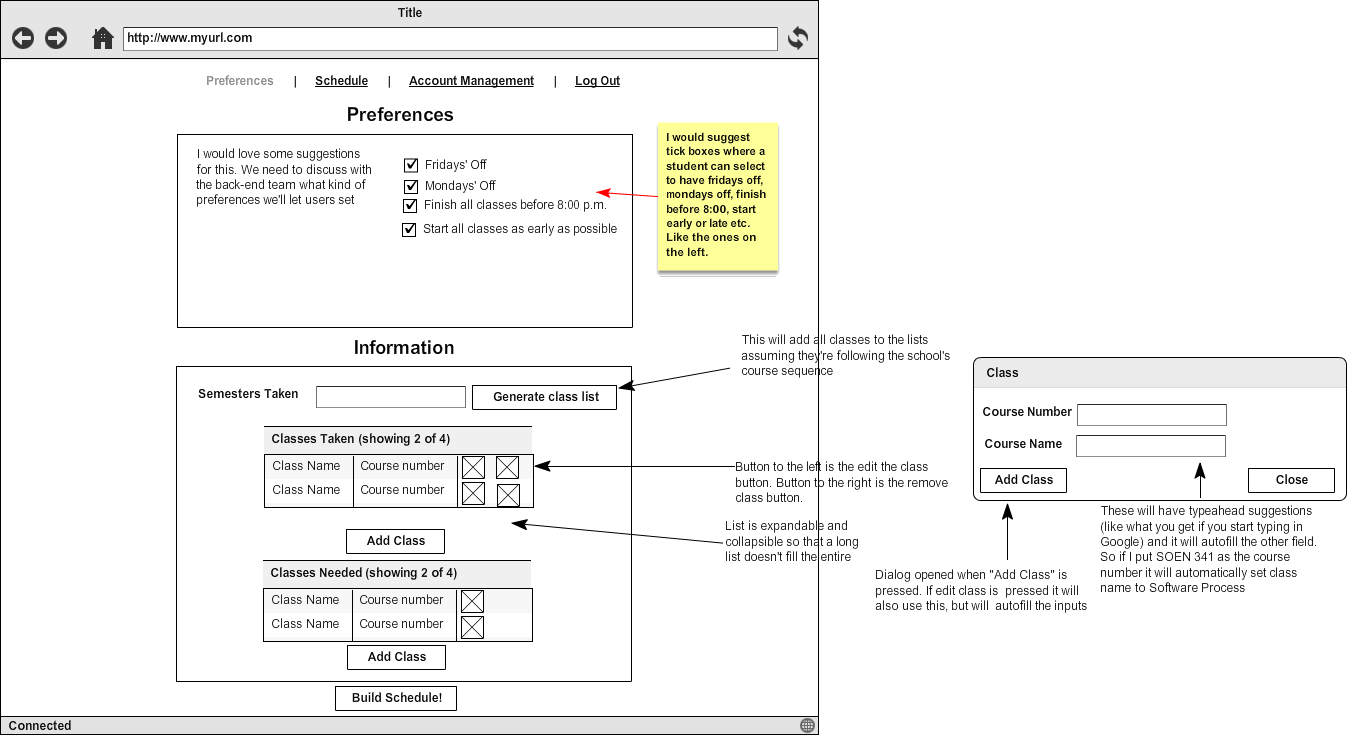


8. Prototyping

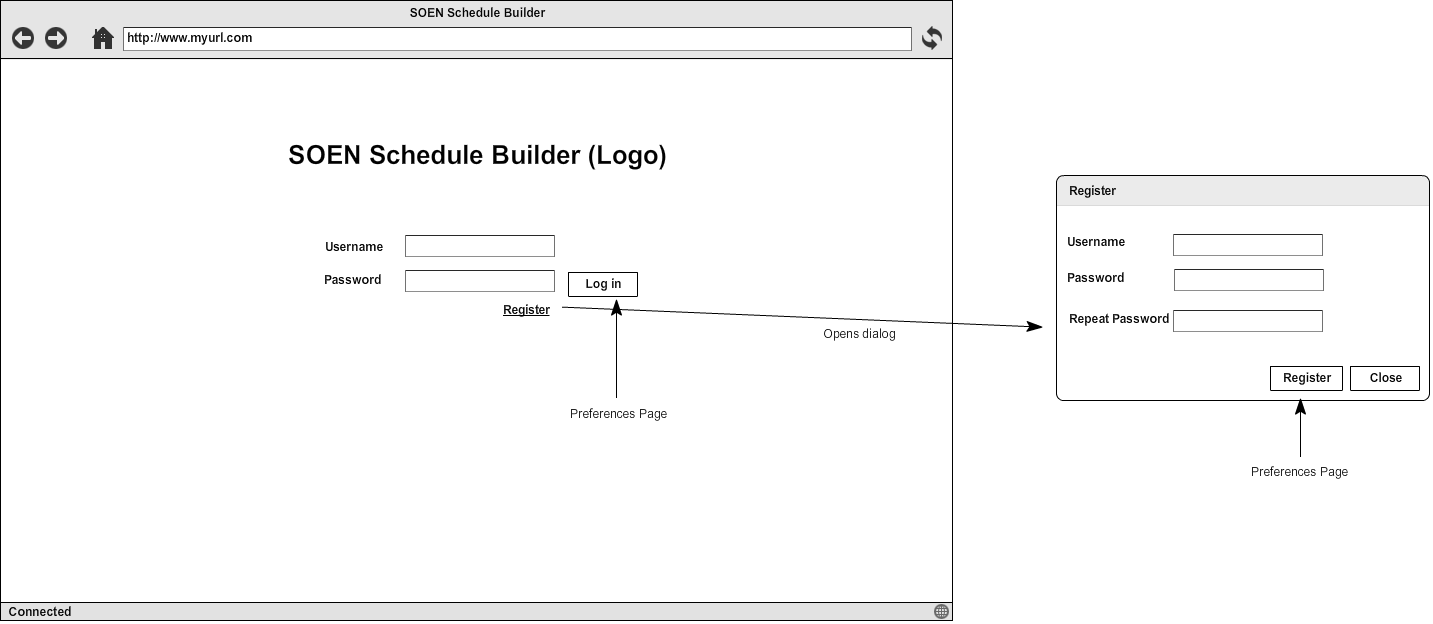
8.1 Schedule Page



8.2 Preference Page



8.3 Home Page



8.4 Account Manage Page

