**Senior Project Website**

**Version 5**

**Design Document**

CIS 4911 Senior Project

Section U01

Professor Masoud Sadjadi

**Prepared by**

Jacek Kopczynski

Yamel Peraza

April 23th, 2015

# 

Copyright

*All title, including but not limited to copyrights, in and to the SENIOR PROJECT WEBSITE are owned by School of Computing and Information Sciences Faculty of Florida International University. All title and intellectual property rights in and to the content which may be accessed through use of the SENIOR PROJECT WEBSITE is the property of the respective content owner and may be protected by applicable copyright or other intellectual property laws and treaties. This EULA grants you no rights to use such content. All rights not expressly granted are reserved by School of Computing and Information Sciences Faculty of Florida International University.*



Abstract

*This document covers the design of the solution to the requirements analyzed in the requirement analysis page. The design of the solution includes an introduction to the overall document and project. Overall system design such as hardware/software mapping. Detailed design of particular subsystems. And an appendix of related information.*

Table of Contents

*Senior Project Website V5*



[Copyright 2](#_Toc393972296)

[Abstract 3](#_Toc393972297)

[Table of Contents 4](#_Toc393972298)

[**1.** **Introduction** 5](#_Toc393972299)

[**1.1.** **Problem Definition** 5](#_Toc393972300)

[**1.2.** **Design Methodology** 6](#_Toc393972301)

[**1.3.** **Terminology – Definitions, acronyms, and abbreviations** 7](#_Toc393972302)

[**1.4.** **Overview of Document** 9](#_Toc393972312)

[2. System Design 10](#_Toc393972313)

[**2.1.** **Overview** 10](#_Toc393972314)

[**2.2.** **Subsystem Decomposition** 10](#_Toc393972315)

[**2.3.** **Hardware and Software Mapping** 14](#_Toc393972316)

[**2.4.** **Persistent Data Management** 14](#_Toc393972317)

[**2.5.** **Security/Privacy** 19](#_Toc393972318)

[3. Detailed Design 22](#_Toc393972319)

[4. Glossary 71](#_Toc393972343)

[5. Appendix 73](#_Toc393972353)

[**5.1 Appendix A Used Case Diagrams (SPWv.5)** 73](#_Toc393972354)

[**5.2. Appendix A Used Cases (SPWv.5)** 73](#_Toc393972356)

[**5.3 Appendix C Class Interfaces (SPWv.5)** 76](#_Toc393972357)

[**5.4 Appendix D Diary of Meetings and Tasks** 79](#_Toc393972406)

[6. References 80](#_Toc393972407)

1. **Introduction**

*Senior Project Website V5*



This section introduces the overall SPWv.5 product. Starting by defining the problem, then the design methodology used in implementing the product’s design, followed by various terms used throughout the document, and finally a brief overview over all chapters of this document.

* 1. **Problem Definition**

For Senior Project class, students are required to use virtual machines to develop and deploy their applications. The process of requesting virtual environments currently is done manually, given that Senior Project Website does not have the functionality of processing or managing virtual machine requests for its users. The impractical approach of manually finding each student’s email requesting virtual machines is time-consuming, lacks efficiency and is error prone.

The head professor would like to extend the usability of the system by allowing students to request and customize their virtual machine settings, while letting him manage the students’ requests.

We also plan to extend the functionality of the site by adding smaller features like: forgot password, adding students with incomplete grades, changing passwords through act as user, and fixing problems with functionality of the old site.

* 1. **Design Methodology**

The SPWv.5 follows a new software development model different from prior ones. The development model used this semester was based on agile methodologies. Small incremental developments based on weekly progress resulted in the virtual machine subsystem released in the end.

Daily and weekly meetings resulted in discussions regarding what was done since the previous meeting, the plans for the following week, and scaling of expectations for the final developmental increments. The benefit of this system lies in its freedom to deploy development on a regular basis for feedback and tweaks. This is a step forward from the waterfall method, where much more analysis happens with a rush of development in one period of time. The main drawback of the agile methodology is that documentation related to the process has less time to be compiled and changes are not fully analyzed to extreme depths as in the waterfall method. Nevertheless, documentation is compiled for the benefit of future developers to highlight changes and to greatly elaborate on the project’s status.

* 1. **Terminology – Definitions, acronyms, and abbreviations**

EULA: End-User License Agreement

FIU: Florida International University

Google Docs: Free web-based office suite offered by Google within Google Drive service.

PHP: Open source server-side scripting language designed for web development to produce dynamic web pages.

SPW: Senior Project Website

SPWv.1: Senior Project Website Version 1

SPWv.2: Senior Project Website Version 2

SPWv.3: Senior Project Website Version 3

SPWv.4: Senior Project Website Version 4

SPWv.5: Senior Project Website Version 5

**NRMP**: A form of matchmaking devised by the national residency matchmaking program (NRMP). The details of it are that hospitals have residency spots to fill and a ranked list of applicants they want filling them. Applicant ranked residency programs themselves. The matchmaking process works by having applicants apply to residencies in their list’s order. If another applicant tries to displace a current tentative applicant they displace the tentative applicant if the challenging applicant is more desired by the program else they try the next program, displaced applicants try their next program too. Matching ends when all applicants are match, or all applicants are match minus ones who went through their entire list.

**Heuristics**: Colloquially means “common sense approach”, in computer science the meaning is adapted to mean a technique applied to solve a problem. I.e. to shorten average job time in a computer do short jobs first.

**VIP**: Very important project, a project ranked by the head professor to be of a score between 2 and 100 this project will undergo intense matchmaking using heuristics to find a perfect team quickly (higher scored means higher priority)

**OP**: Other project, a project ranked by the head professor to have a score of 1, such projects will undergo NRMP matchmaking to give the ability to compromised project proposer and student constraints. These are more hands-off.

**LinkedIn:** Social media website for business professions to network and advertise themselves

* 1. **Overview of Document**

**Section 1:** Introduces the project as a whole as well as specific document introduction.

**Section 2:** Covers overall system design including subsystems, hardware/software making, and how data and security/privacy are addressed.

**Section 3:** The detailed design goes into more details regarding affected subsystems utilizing static and dynamic diagrams as well as code specifications.

**Section 4:** Contains a glossary of terms.

**Section 5:** The appendix contains more detailed information for prior sections as well as from prior documentation.

**Section 6:** Contains references utilized in the document.

1. System Design

*Senior Project Website V5*



This section will expand on the details of the system’s overall design. It will give the reader a more thorough overview of the system and subsystem decomposition, how hardware and software is mapped, the manner that persistent data is defined, and how security and privacy is utilized.

* 1. **Overview**

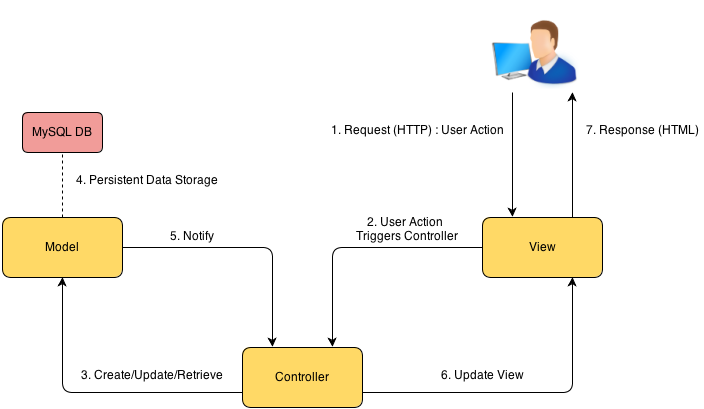
The SPW’s system is based upon the Three-Tier and MVC architectures. It is further decomposed into 7 subsystems: virtual machine, matchmaking, login, project, user management, repository, and Rest API. The main objective of SPWv.5 was tuning the existing functionality of the system plus add the new virtual machine subsystem.

For deployment in hardware and software, the architecture is based on client and server, wherein a browser (client) accesses the SPW application (on a server). In addition, the API works as an API for other senior projects requiring data on registered users.

The persistent data change with respect to the previous one (SPWv.4). In SPWv.5 a total of three new tables were added to handle the virtual machine requests, virtual machine images, and the default school’s system admin info. The rest of the tables remained the same, containing tables for users, language, projects, experience, skills, milestones, rankings, etc. Security and privacy features will remain the same as in SPWv.4. Having a one-way password encryption, session encryption, etc.

* 1. **Subsystem Decomposition**

The current system (SPWv.5) follows the system architecture of Three-Tier and MVC as a means to define how the system is structured. This structure continues being the same for SPWv.5. The style fits perfectly the function of the SPW, controllers maintain business logic for interactions, views allow users to observe relevant data, and models hold the means to access data from the database. For a user, that flow of logic is functional. Their interactions on the views prompts the controller to respond, and the controller requests database data from models when required. From a three-tier perspective, the user’s browser and what they see (the view) is the presentation tier, the controller’s reaction to the user’s interaction is the logic tier, and the controllers request for data from the model is the data tier. Refer to picture below for a graph describing the nature of the system from SPWv.4 design document.



The SPW is further decomposed to 7 subsystems. The virtual machine subsystem the most recently added therefore it will be discussed first. For the purposes of documentation, the remaining 6 subsystems will be described adapted from what was said in SPWv.4 design document.

**Virtual Machine Subsystem**: The subsystem will handle all the necessary logic for the request of virtual machines (creating virtual machine request, modifying virtual machine request, deleting virtual machine request, filtering virtual machine request, etc.). Allowing students to customize their virtual machines needs and giving to the head professor the means to manage students’ virtual requests.

**Matching Algorithm Subsystem:** This subsystem will facilitate the matching of students to projects by the head professor. It will function in a two-phase matchmaking system where the head professor’s very important projects are filled up first taking into account student interest and the head professor’s interest, with greater emphasis on the latter’s influence. The second phase emphasizes the student’s interest in projects the head professor has little need to put emphasis on. These will be match via NRMP with customization with regards to focus on student interest only or compromise with project’s need for optimization.

**Login Subsystem:** This subsystem will deal with all the logic involving the authentication of the users who use SPW.

**Projects Subsystem:** This subsystem will deal with all the logic that required in relation to projects. Such as proposing a project.

**User Management Subsystem:** This subsystem will deal with user management by the head professors (adding, deleting users, acting as users, etc). Also includes functionality such as synchronizing with LinkedIn, changing passwords, functionality involving the changing of a user’s state essentially.

**Repository Subsystem:** This subsystem will handle all the logic necessary to establish the project files repository functionality. Deals with file upload for projects and milestones (goal for project), allow team sharing, and allows the head professor to organize the structure of the repository.

**Rest API Subsystem:** The SPW will be accessing an external API that is going to be developed as a RESTful service. In essence, this API will be used to verify that a student who wishes to register with the SPW is indeed a student that is currently enrolled in the Senior Project course. The API was developed as part of the SPWv.2 project and only allows students that are currently enrolled in the senior project course to login as students. Without this API, any student with an FIU email account would be allowed access to the system. It is important to note that other services (developed by other senior project teams) will also access this API in order to conduct validation of their student users. Of note Jonathan Santiago of the Collaborative Platform team of Summer 2014 did work on this subsystem for SPWv.4 as it was broken and not functional.

* 1. **Hardware and Software Mapping**

This aspect for SPWv.5 is largely unchanged from the description of SPWv.4 in their design document. The hardware and software are essentially based on three architectural patterns. In a commonplace client-server architecture, the client uses a web browser to access the server running the SPW application. This application interacts with a MySQL server for database needs and runs under an Apache Tomcat server.

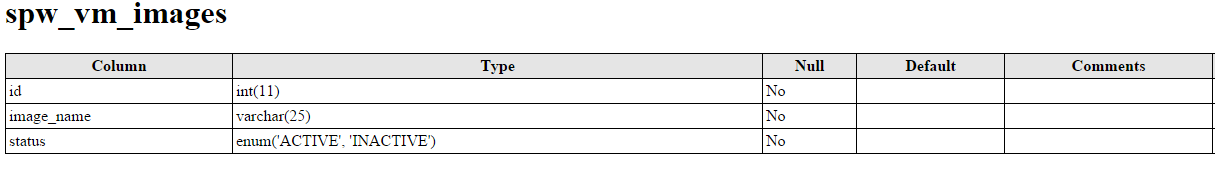
Additionally the server will run an API to verify members in the SPW amongst other projects in the senior project class. Below is a display of this from SPWv.3 design document.

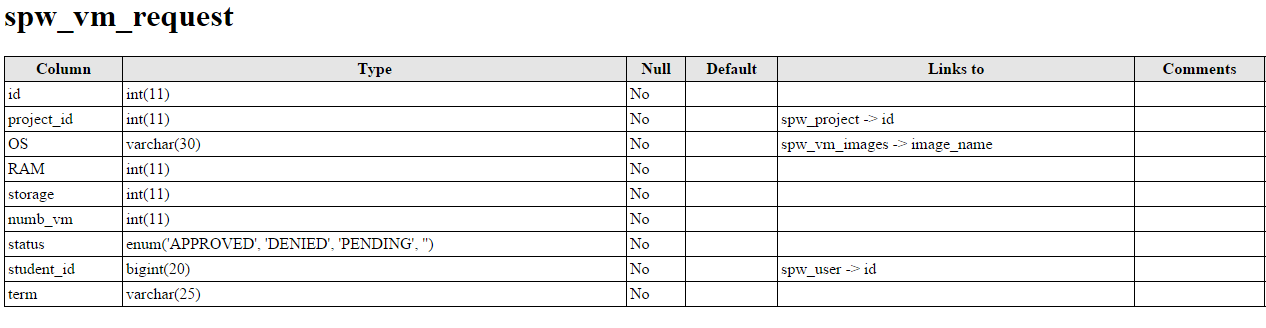
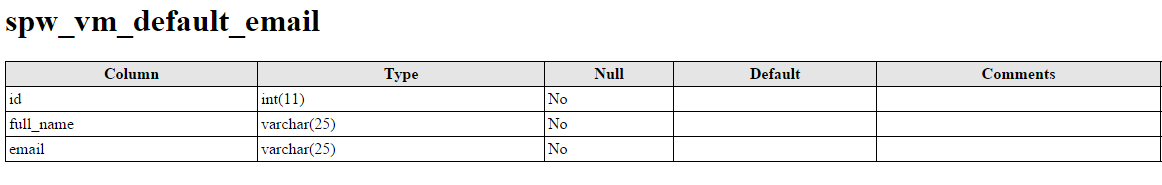


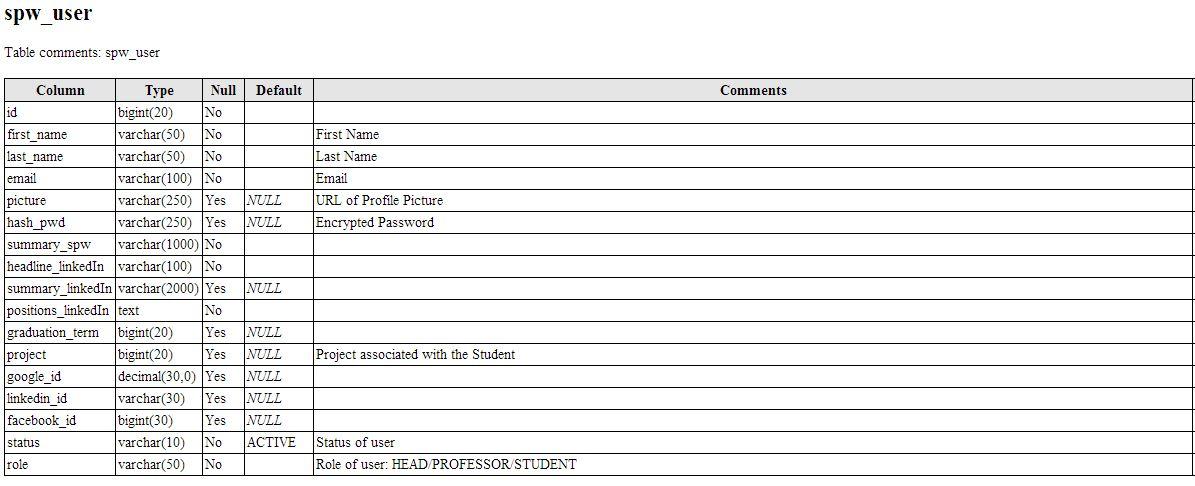
* 1. **Persistent Data Management**

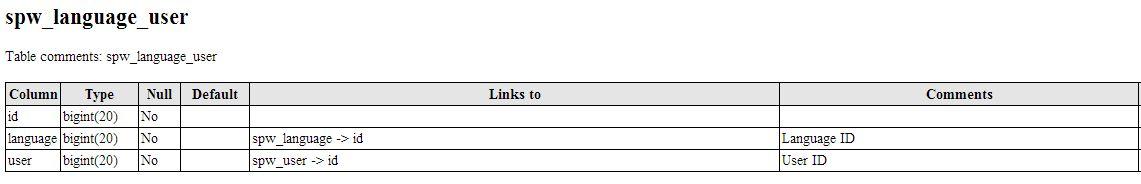
The persistent data did change from SPWv.4 to SPWv.5. Three new tables were added into the system, the rest of the tables were kept as from previous version SPWv.4.

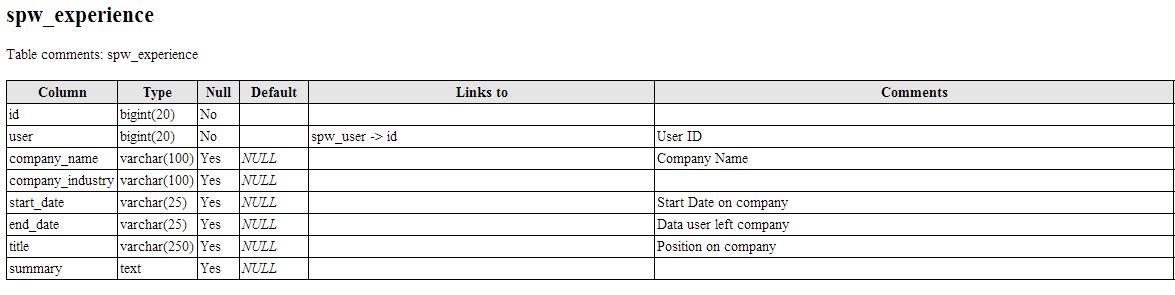
**Data Dictionary for SPW (from SPWv.4 design document):**

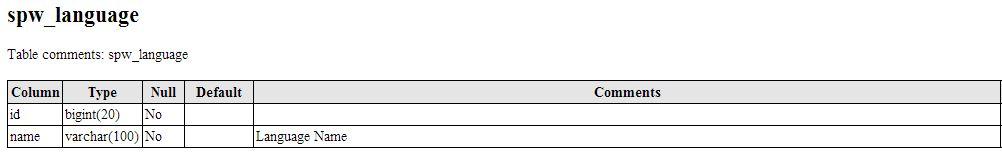


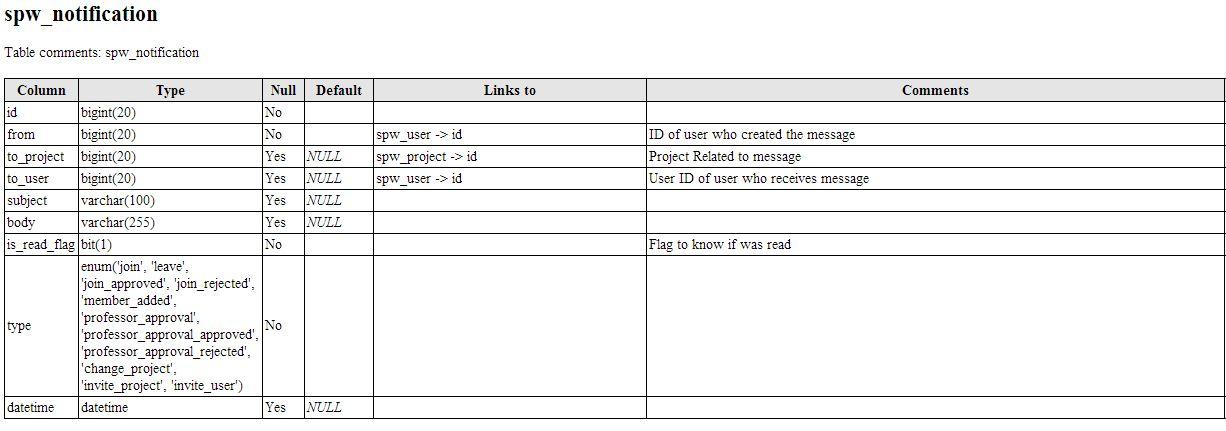


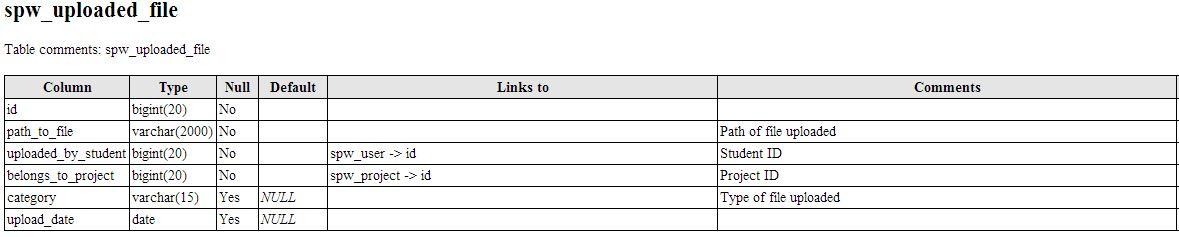
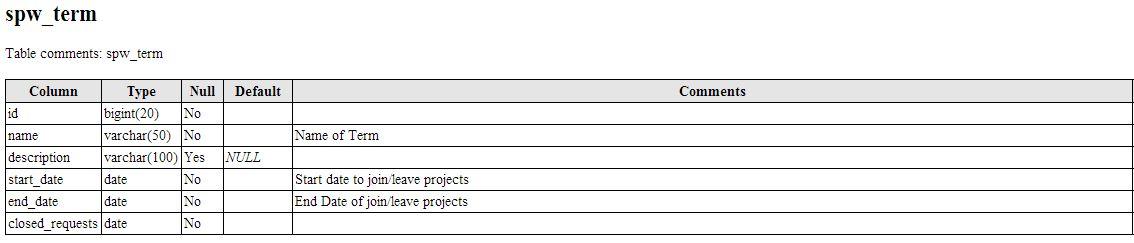
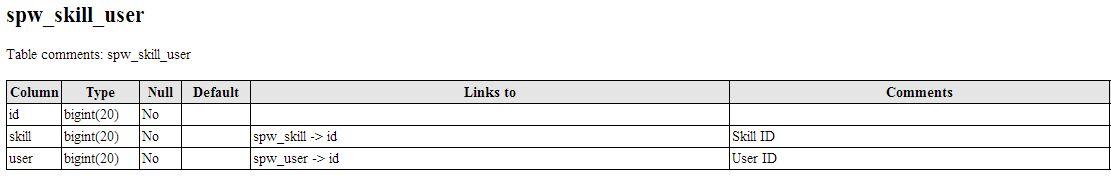
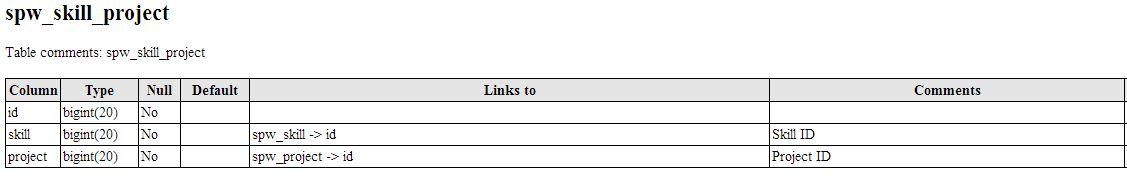
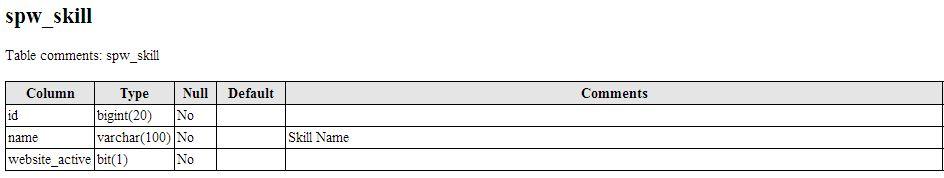
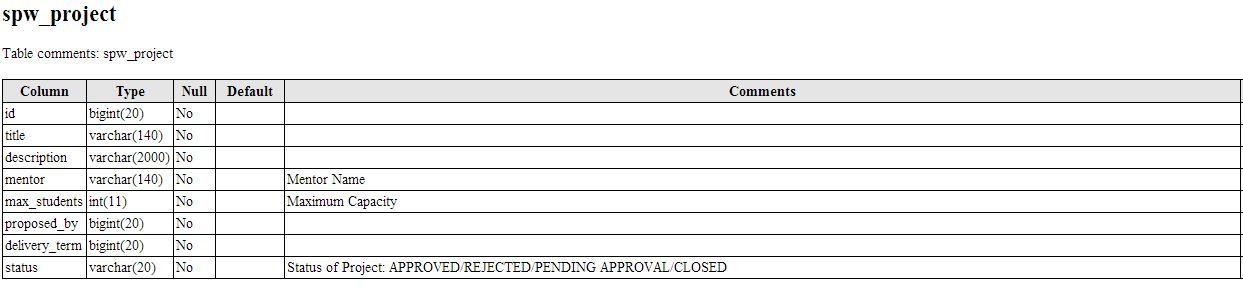


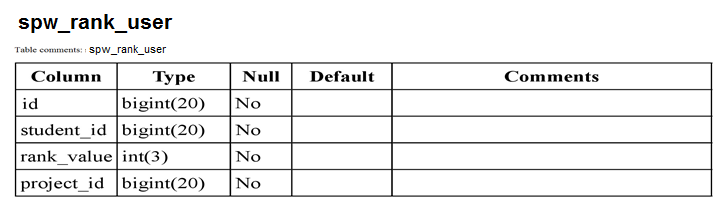


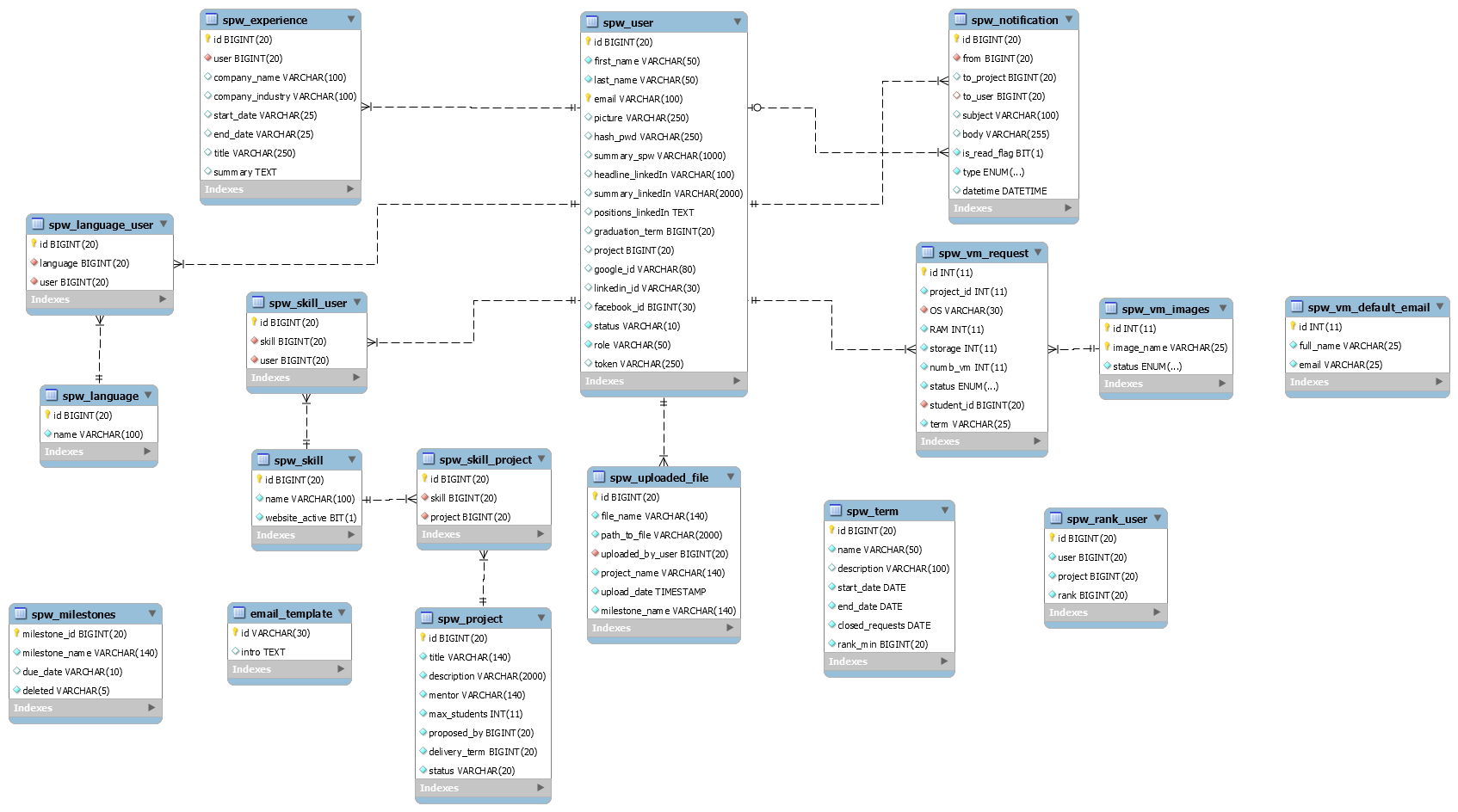










****

Entity-Relationship Diagram for Model Subsystem

* 1. **Security/Privacy**

No changes to any of the following occurs going from SPWv.4 to SPWv.5. The following information is entirely from SPWv.4 design document.

SQL Injection

System takes care of avoiding SQL injection attacks by using the Active Record class from the Code Igniter framework for all interaction with SPW Website, which produces safer queries since the values are escaped automatically by the system.

URI Security

Malicious data can be passed to the application via the URI strings. This is prevented in our system by the use of the Code Igniter framework which is fairly restrictive regarding which characters it allows in the URI strings.

Cross-site scripting (XSS)

Cross scripting is also prevented by the use of the Code Igniter Cross Site Scripting Hack prevention filter which has been configured to run automatically to filter all POST and COOKIE data that is encountered.

Cross-site request forgery (CSRF)

Cross site forgery has been also addressed by enabling the CSRF protection on Code Igniter, which turns on a complicated algorithm that allow SPW website to know if the data it receives actually comes from a form on this website, avoiding with it this type of attacks.

Session Encryption

SPW uses the Code Igniter Session class to manage user session information, which is stored as serialized and encrypted in a cookie.

One-Way password encryption

SHA-1 function is used to encrypt users’ passwords in database, which allow website to keep secured user’s password for the website and therefore all their profile information in the website.

The SHA-1 algorithm has two important properties that make of it the more secure algorithm website could use to secure user’s information. The first one is that is a one-way hashing algorithm, which makes impossible to revert back an encrypted output to the initial, plain-text input, and that any given input always maps to the same encrypted value. This ensures that the passwords stored on the server cannot be deciphered by anyone. This way, even if an attacker gains reading permission to the user table, it will do him no good.

It is important to add that no unnecessary information is stored for a user and email addresses are only displayed to registered users.

1. Detailed Design

*Senior Project Website V5*



These following sections cover on greater detail the subsystems described in section 2.

* 1. **Overview**

On this section the Virtual Machine subsystem will be featured with regards to its static and dynamic models of representation; the rest of the models remains the same. Prior document specifications will be added and mentioned as such.

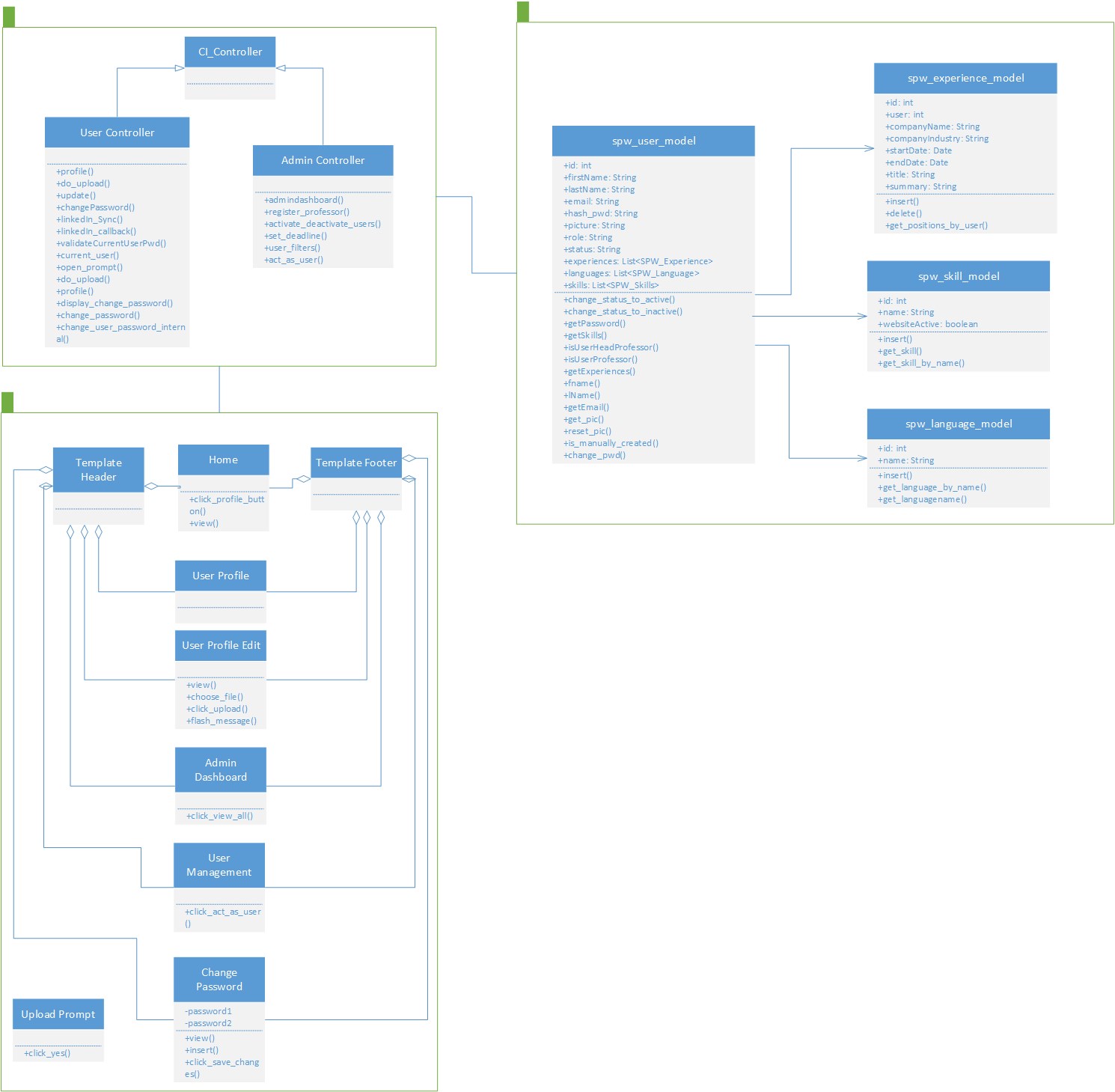
* 1. **Static Model**

A new model and views was added to SPWv.5 which supports the request of virtual machines and virtual machine images to the system. The project controller also had more methods added. What follows is adapted from the SPWv.4 design document and as before, version specific changes are at top of this section and noted as such.

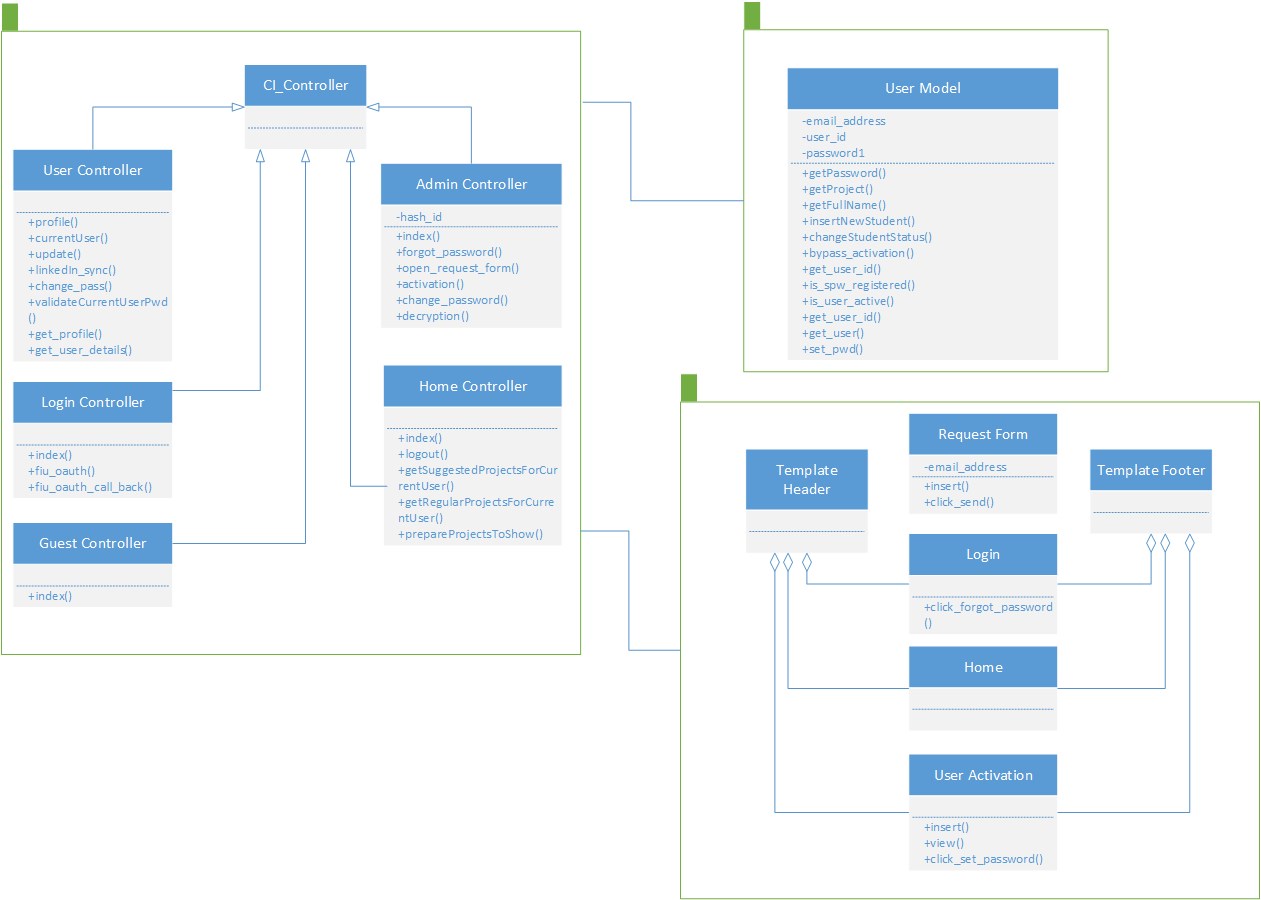
The existing system was built using the Code Igniter framework. These class diagrams display the influence of Code Igniter upon the structure of the model and controller subsystems. Each model and controller defined to be part of the system inherits from CI\_Model and CI\_Controller, respectively. The data that is defined to be part of the each model subsystem is stored in a relational database. An entity relationship diagram shows how the different entities (representative of models) are related to each other in the database schema. The class diagram for the SPW-Register API will also be provided in this subsection. The view subsystem identifies the different views that are part of the system.

SPW makes use of the Front Controller design pattern as it is embedded within Code Igniter. It is used to initialize the base resources needed to run the website. If there are frequent users and they still have a cache file still existent, that cache file will be sent directly to the browser bypassing normal system execution and loading model, helpers, libraries and any other resources required to run the system. The behavior of this pattern is defined in the index.php from Code Igniter framework resources and will act as a main entrance to the webpage.

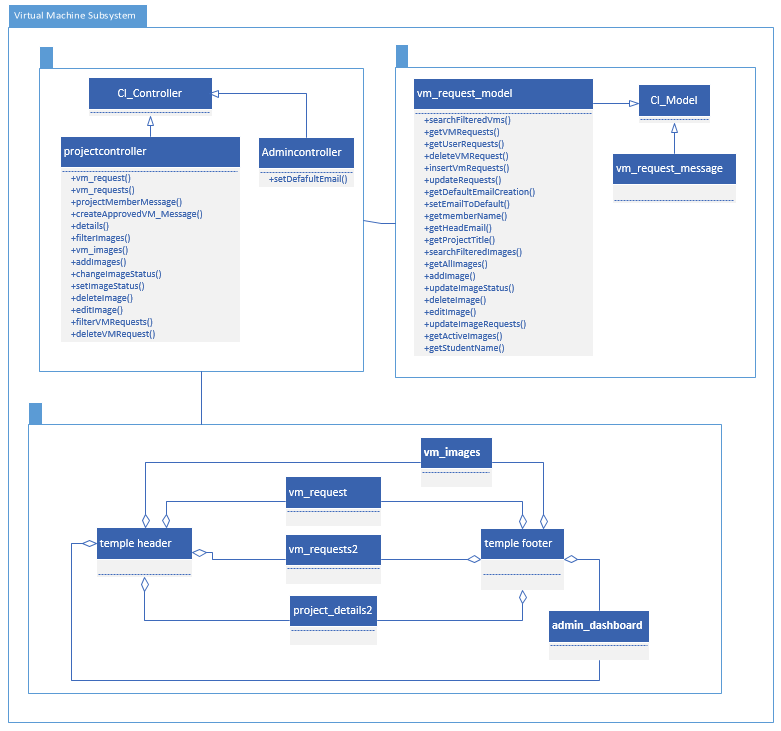
The *Singleton design pattern* is used to define the Term Deadline class. This class represents the available times for students to interact with the projects (propose, join, leave, etc) and it is modified solely by the Head Professor of the class. Only one instance of the class is necessary in the scope of SPW. The pattern ensures that no more than one instance is created at all times.

**Specific to version 5:**

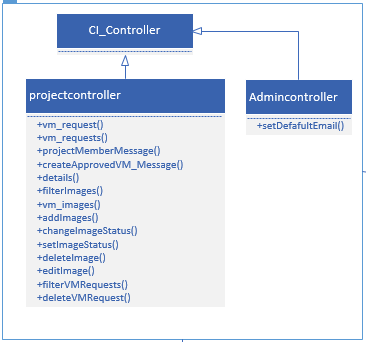
Minimal Class Diagram Profile



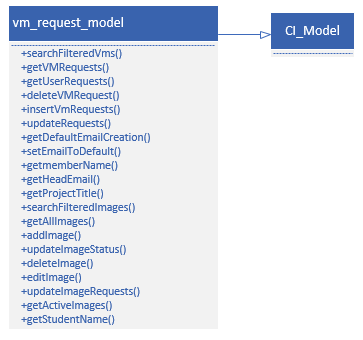
Minimal Class Diagram Logging



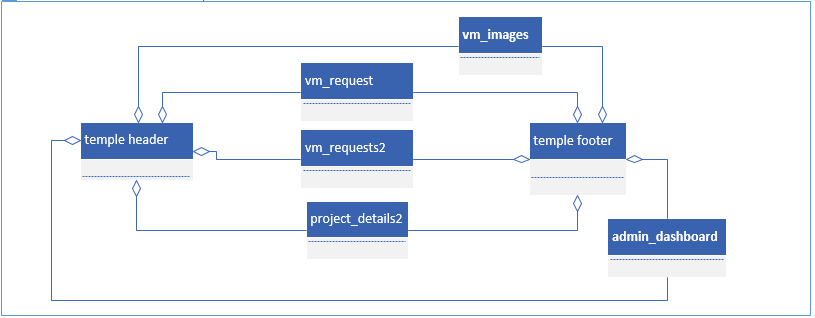
Minimal Class Diagram for Virtual Machine



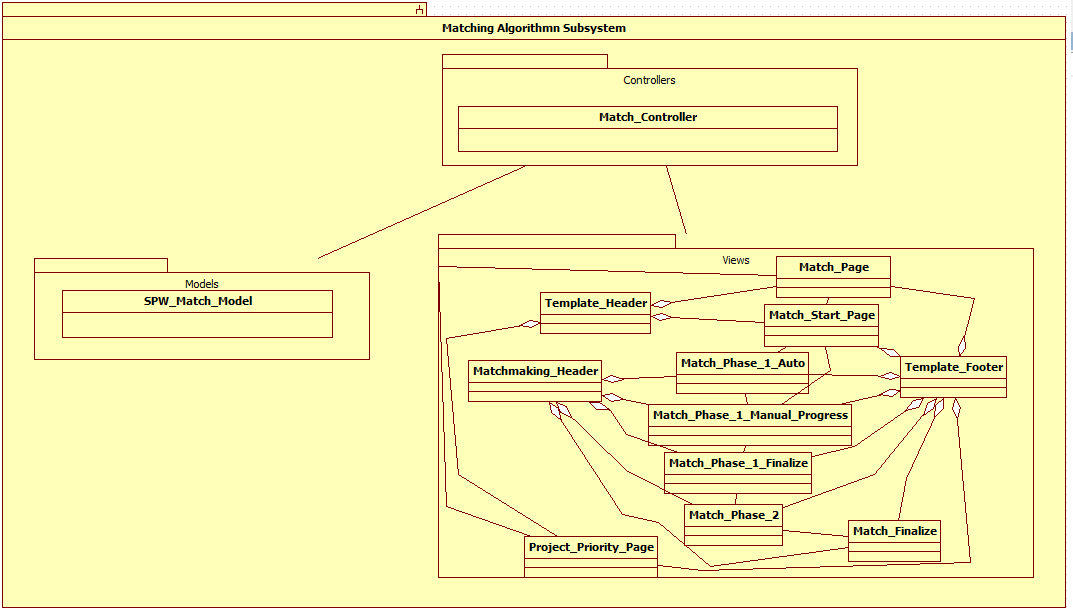
Controller for Virtual Machine



Model for Virtual Machine

View for Virtual Machine

**Specific to version 4:**



Minimal Class Diagram for Matchmaking

****

Controller for matchmaking



Model for matchmaking



Views for Matchmaking

**Specific to version 3:**

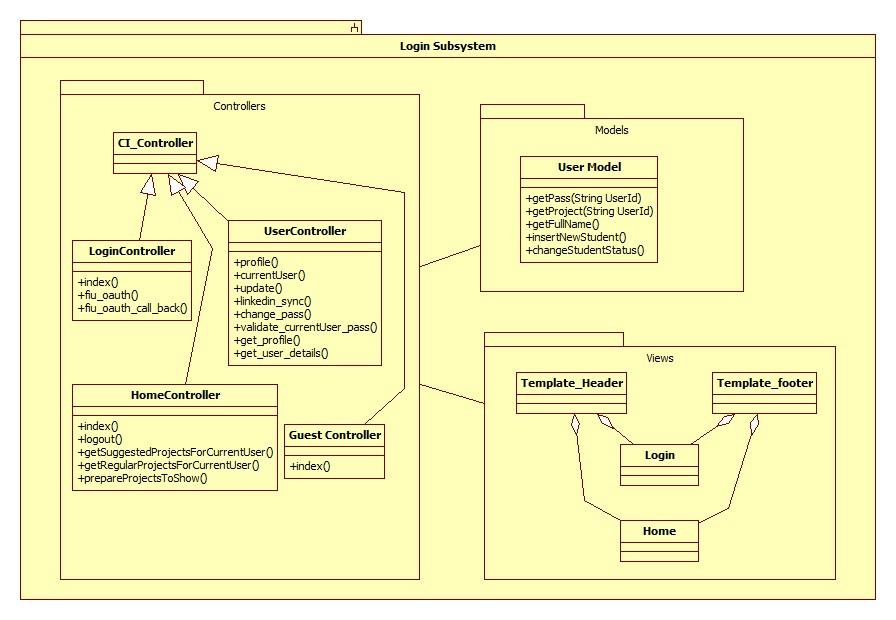
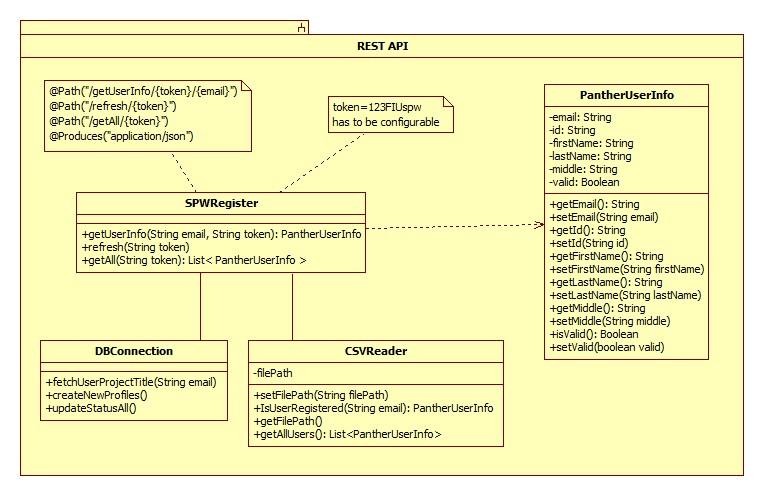
UML Detailed Class Diagrams for the SPWv.3



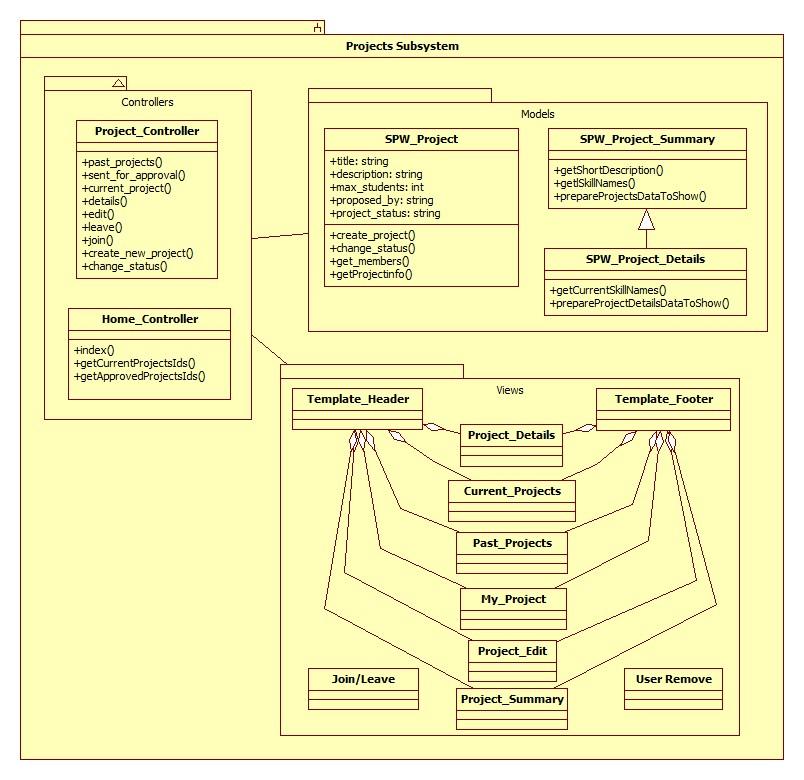




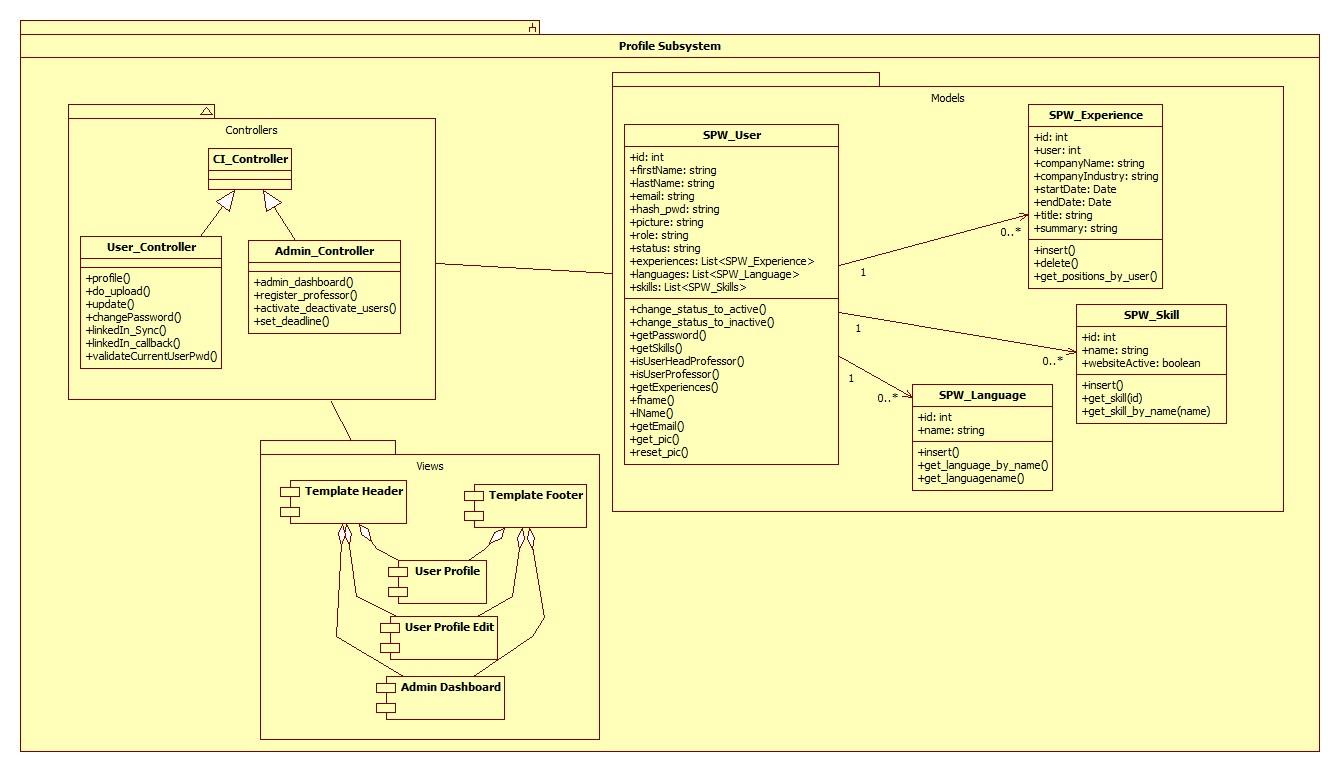
**Specific to version 2:**

 Class Diagram Login Subsystem

Class Diagram for SPW Register API



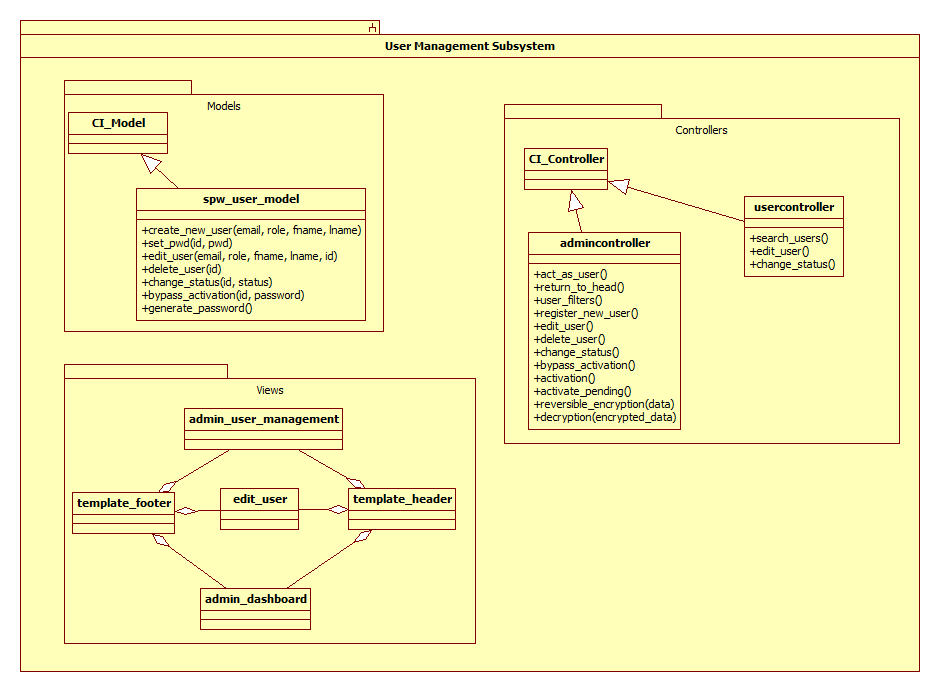
Class Diagram for the Projects Subsystem



Class Diagram for the Profile Subsystem



Class Diagram for the Repository Subsystem



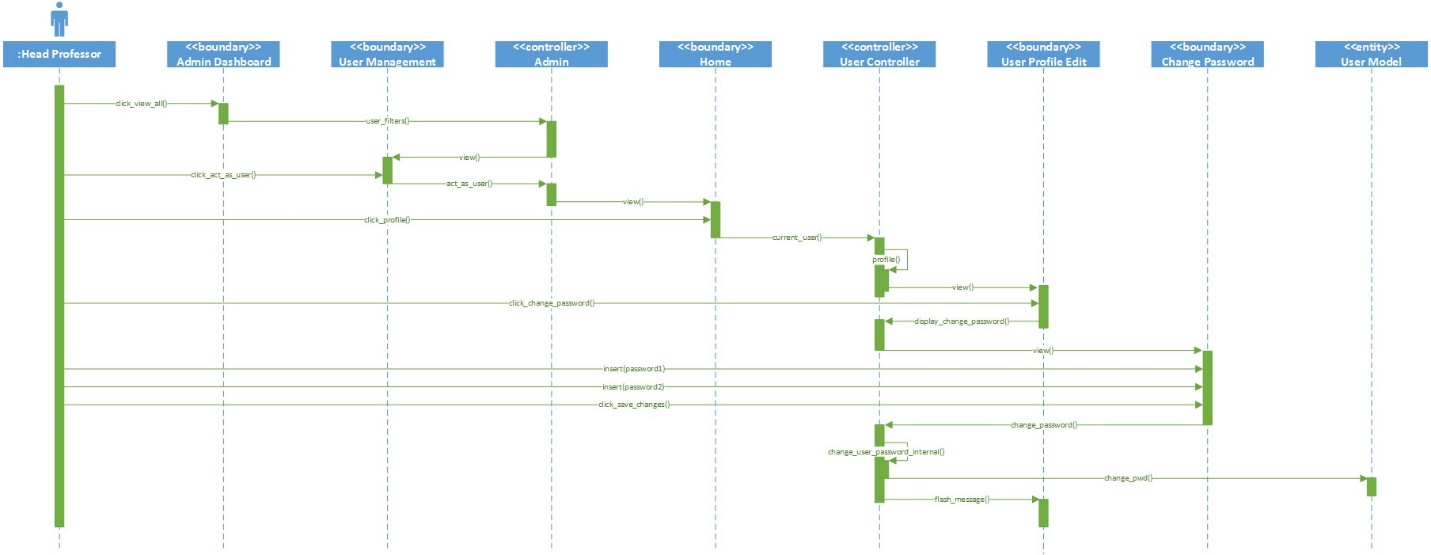
UML Class Diagram for the User Management subsystem

* 1. **Dynamic Model**

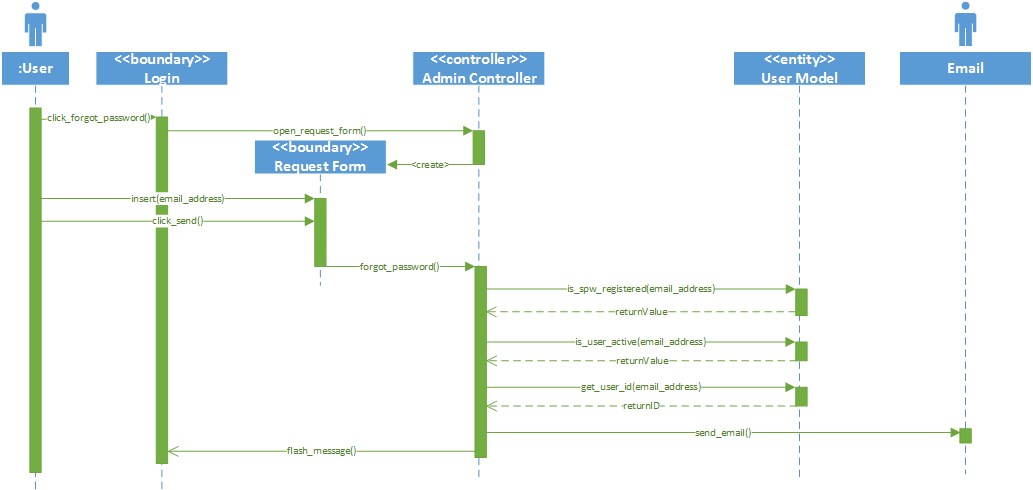
The first part of this section will cover dynamic models specific from SPWv.5, followed by all dynamic models from SPWv.4

**Specific to SPWv.5**

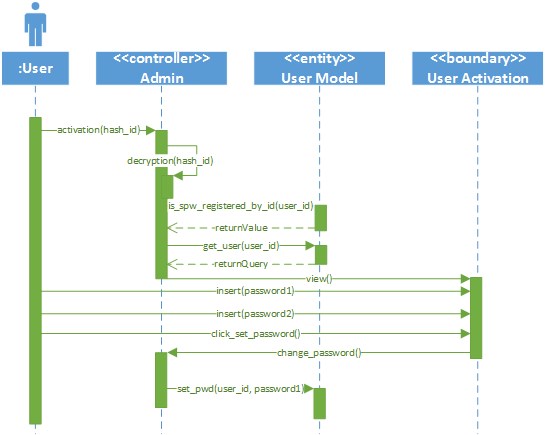
SPW5\_101 Change user’s password



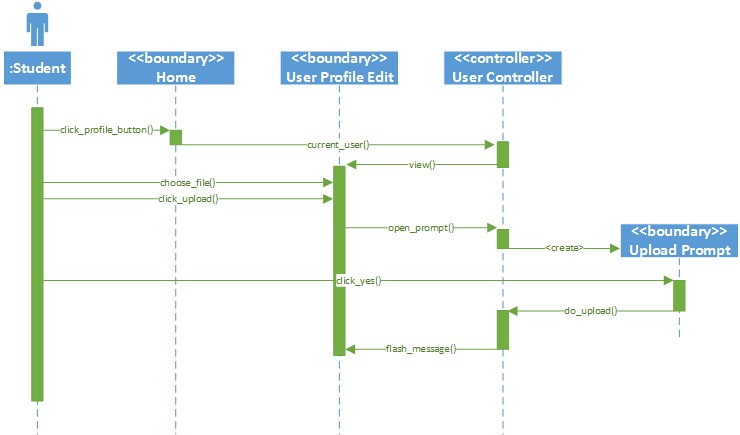
SPW5\_104 Request change of password

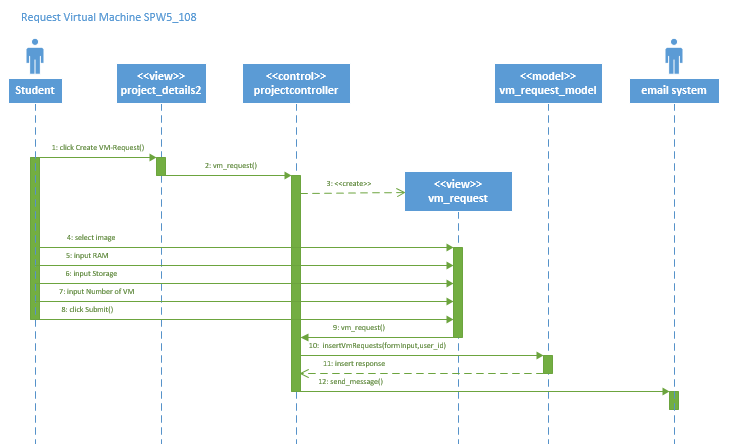


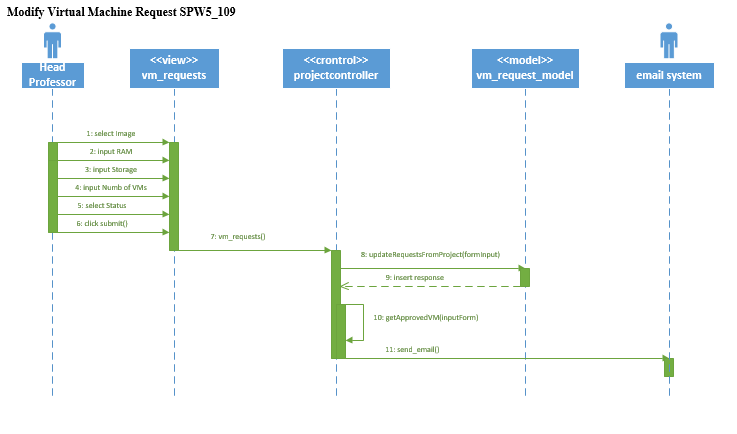
SPW5\_105 Change forgotten password

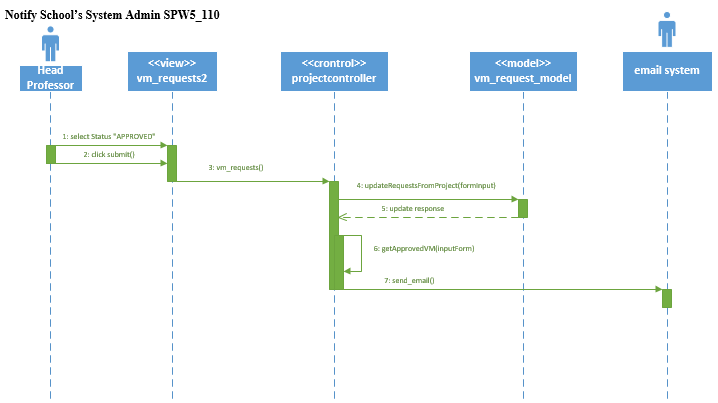


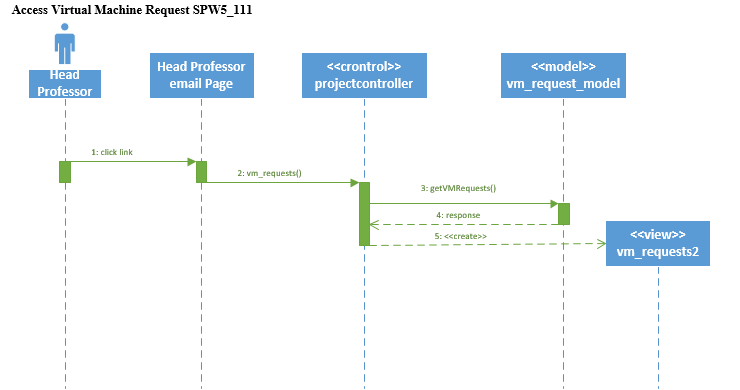
SPW5\_106 Change profile picture

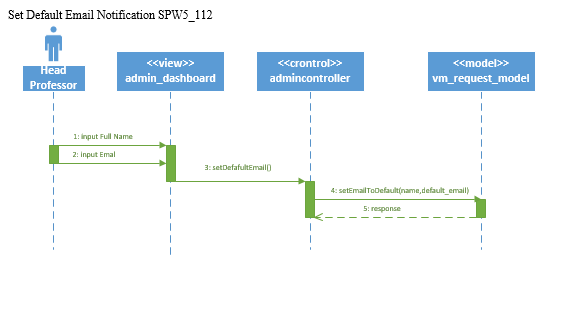


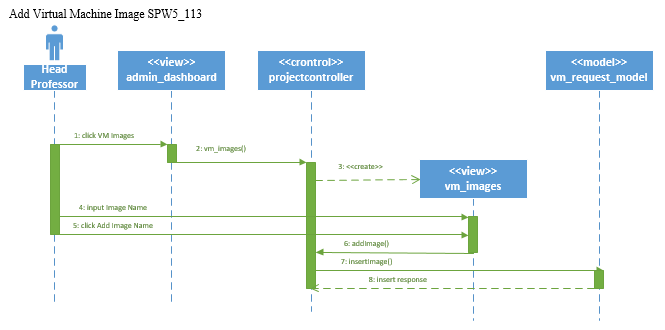


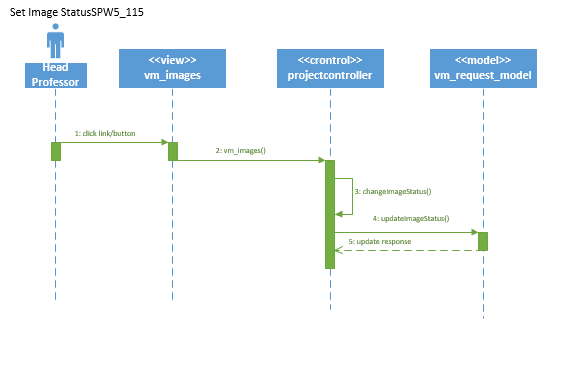


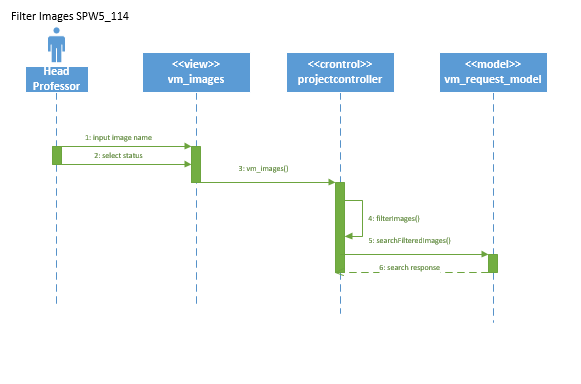


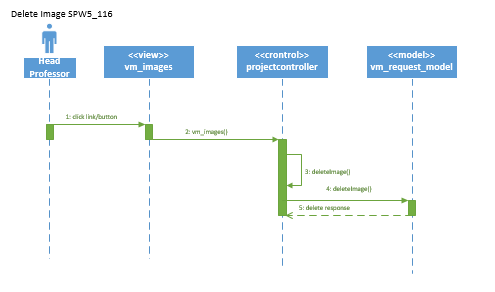


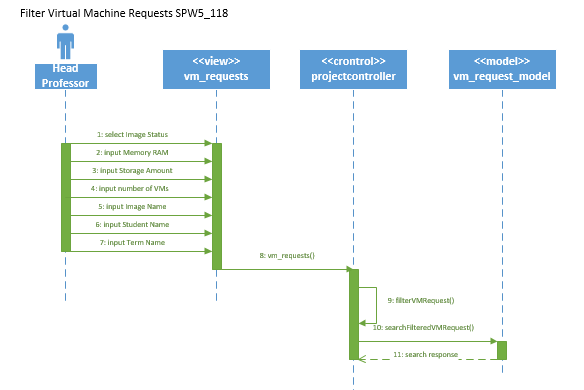


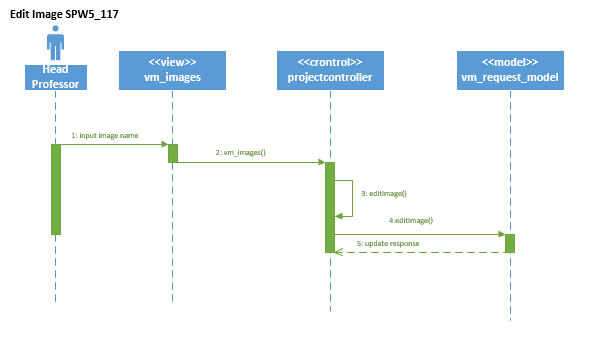


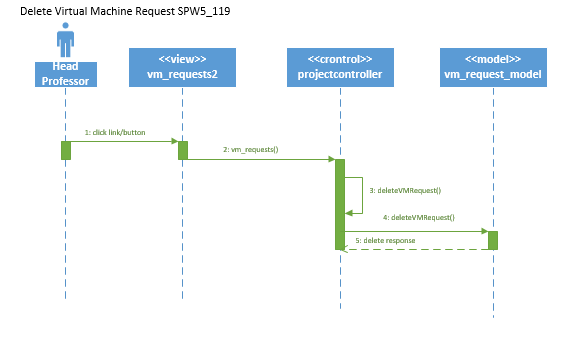




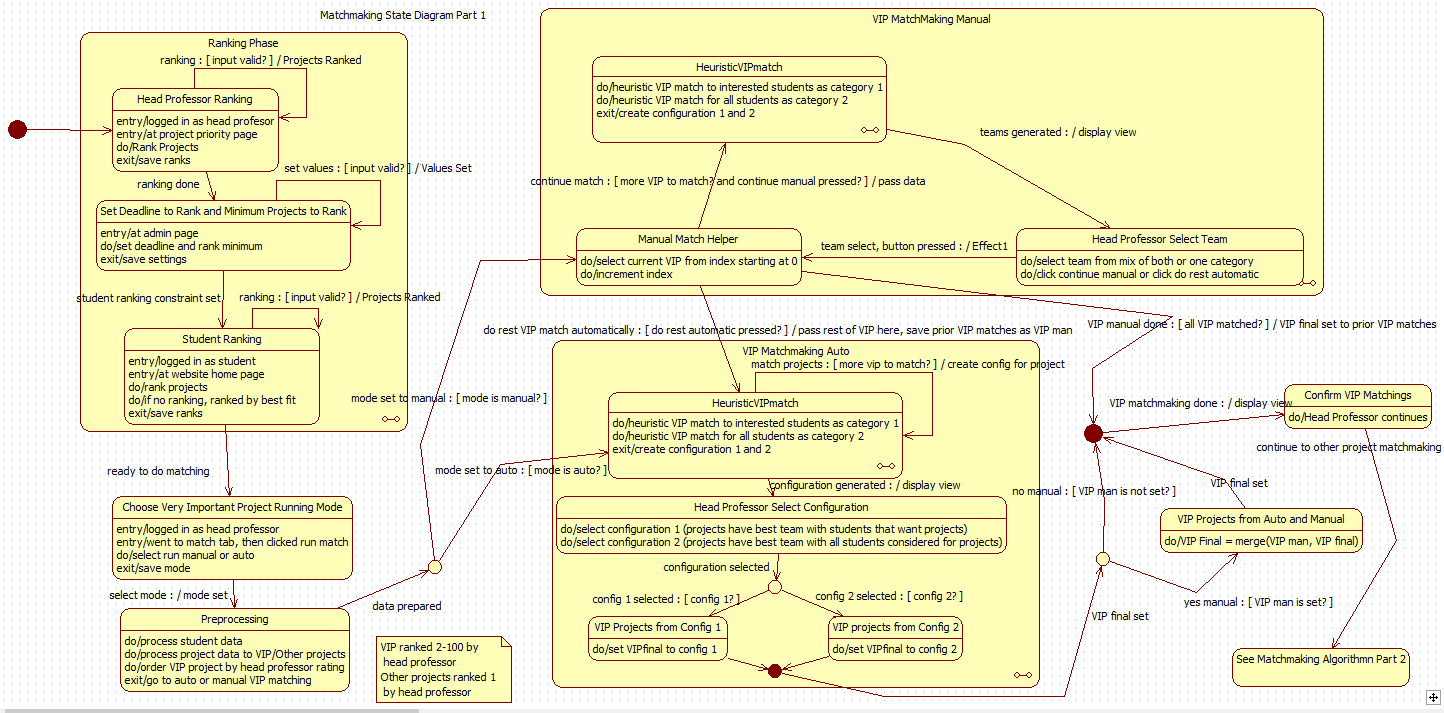


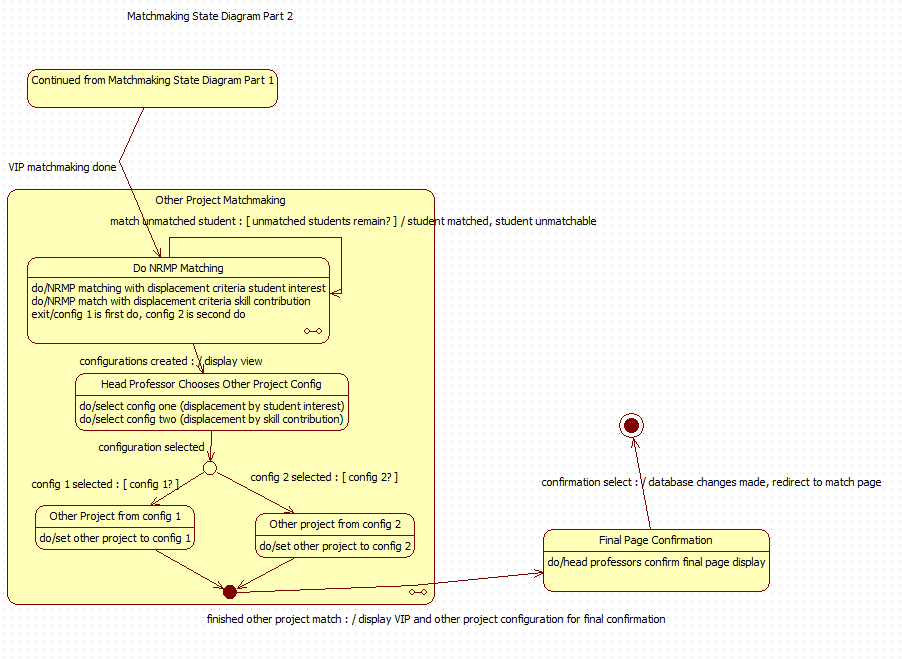


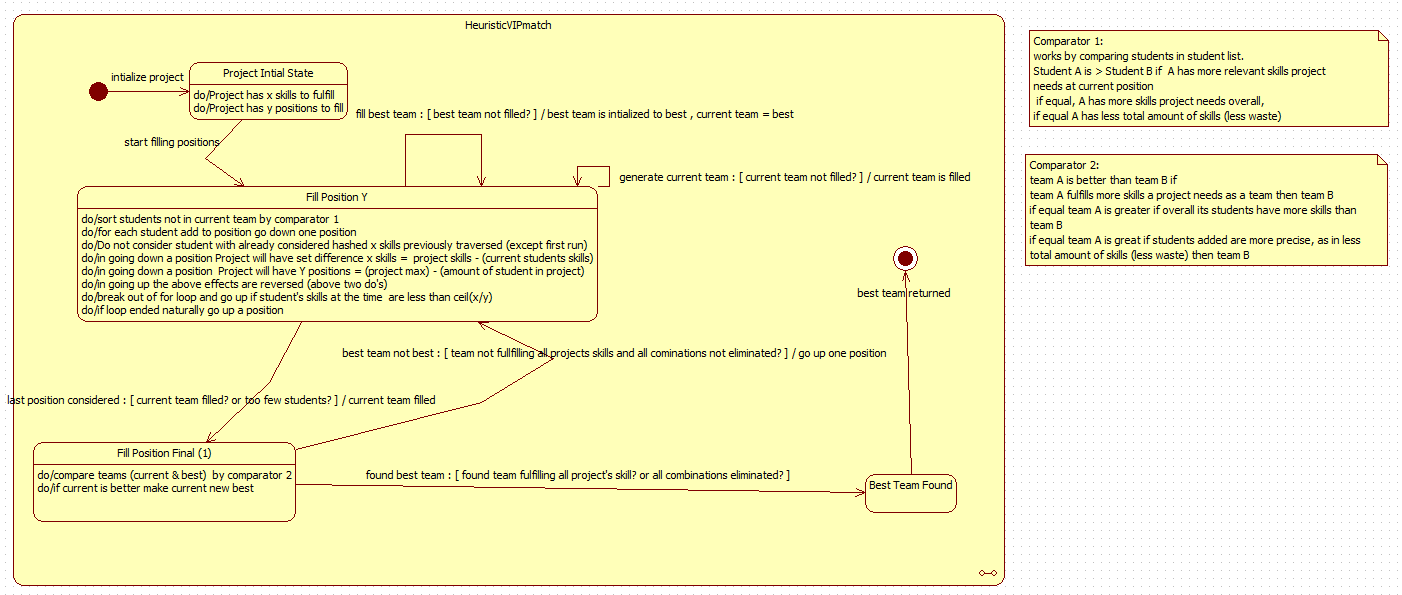


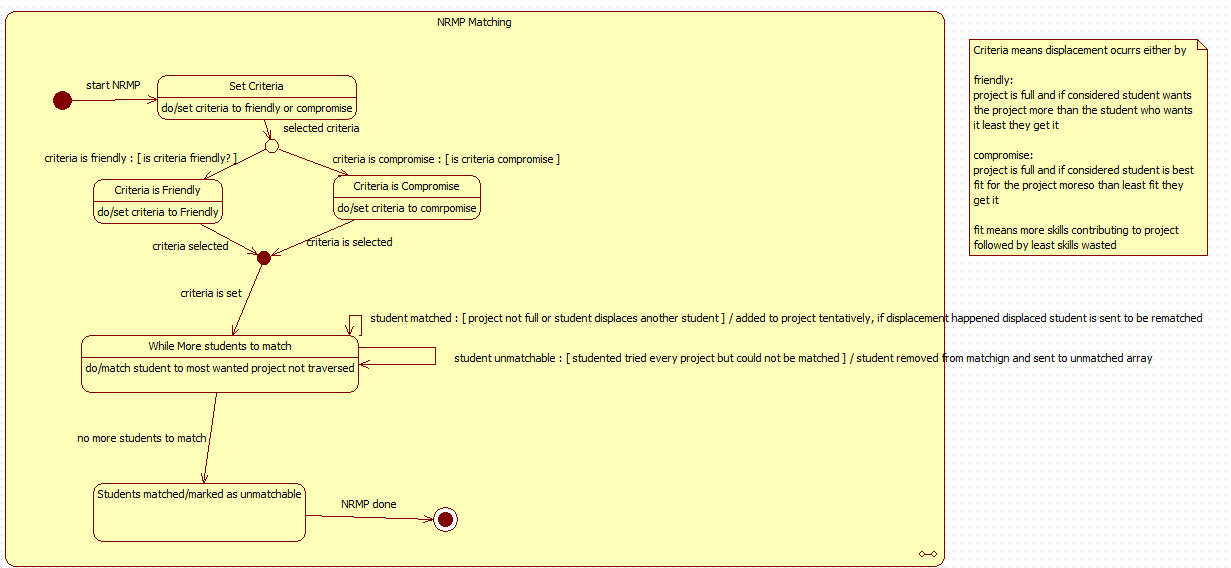


**Specific to SPWv.4:**

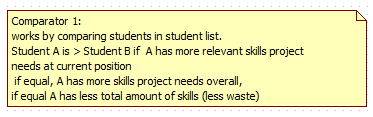
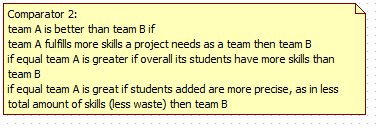
Following two diagrams depict matchmaking state machine.



The following state diagrams depict the states of Heuristic matchmaking and NRMP matching with more details.

****The specific algorithm details for the heuristic VIP are detailed below

1. Given a project with X skill needs and Y positions
2. Go through the Y positions of the projects (going down until all position filled Y==0)
   1. For each student compose relevant skill array (skills project needs at the moment they have)
   2. Sort the student list by comparator 1 see below
   3. For each student
      1. Save relevant skill to has table
      2. If best team not initialized add student to current team and goto to 2. With Y-1 positions and X set difference (Students skills) skills
      3. If student’s relevant skills is less than ceil(count(X)/Y) exit 2.b.
      4. If student’s relevant skill in hash table go to 2.b. to next student
      5. If position Y is not 1, add student to current and goto to 2. With Y-1 positions and X set difference (Students skills) skills
      6. If position is 1, add student to current and compare current to best,
         1. if current better (See comparator 2) and fulfills all project skills exit 2. and best = current
         2. else go to 2.c.
   4. If exit go back to Y+1 2.b. removing latest addition to current
3. Best team is best team



The specific algorithm details for National Residency Matchmaking

1. Select criteria to displace students as friendly (displace student who wants project least with one who wants it most) or compromise (displace student who contributes least to the project least with one who contributes more)
2. For each student unmatched
   1. Go through each project not yet traversed in the order they ranked them
      1. If position open student is matched and removed from unmatched, continue to 2.
      2. Else if student is better than the worst one based on criteria, add student to project, removed from unmatched and put worst student to unmatched, continue to 2.
   2. If student reached here and is unmatched remove from unmatched and put in unmatchable
3. End result students are matched or said to be unmatchable (i.e. their ranking ended with no match for them or too few projects)

Following is sequence diagram for running a match



Sequence diagram for ranking projects for students and head professor. Validation has changed so remains same from SPWv.3 other than validation of results.

Student Save Rank (SPW3\_205)



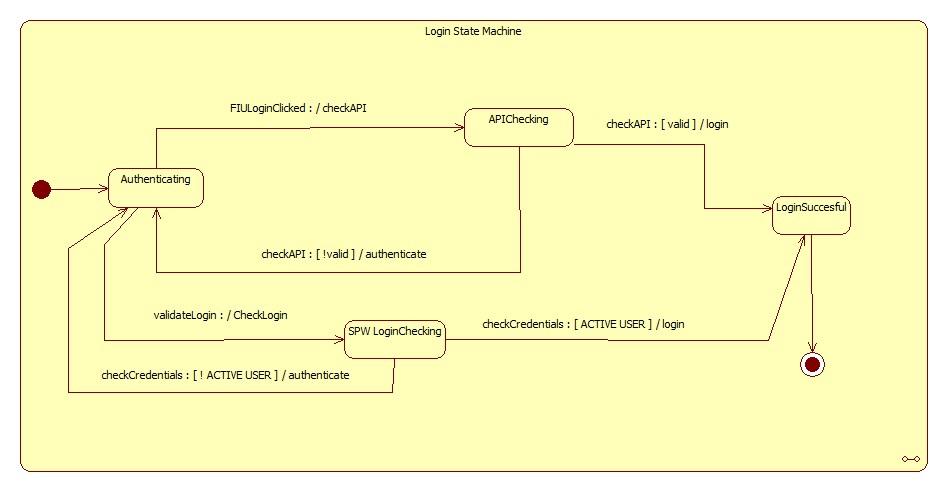
Professor Save Rank (SPW3\_205)

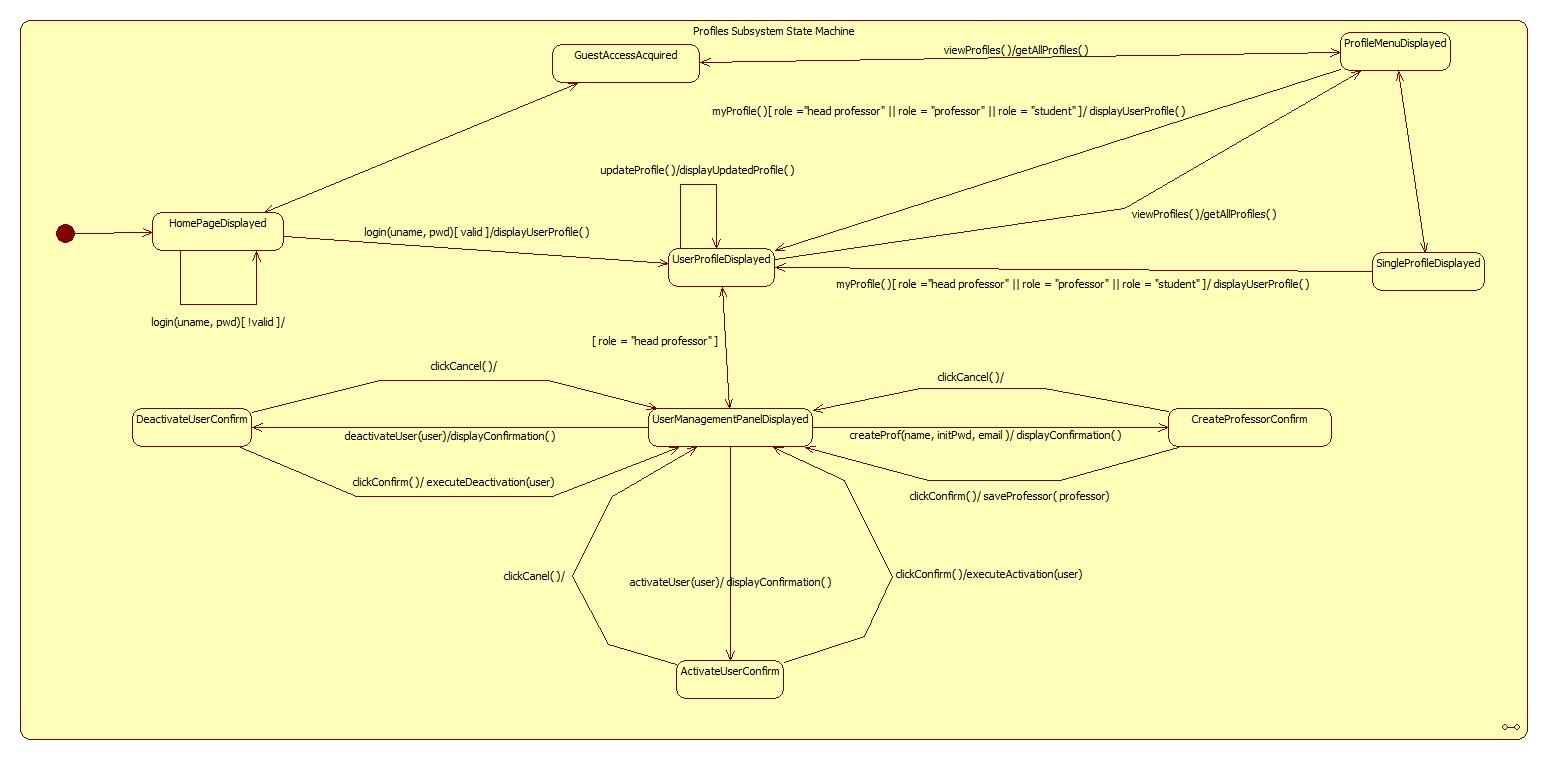
Project Priority Page not Match Page



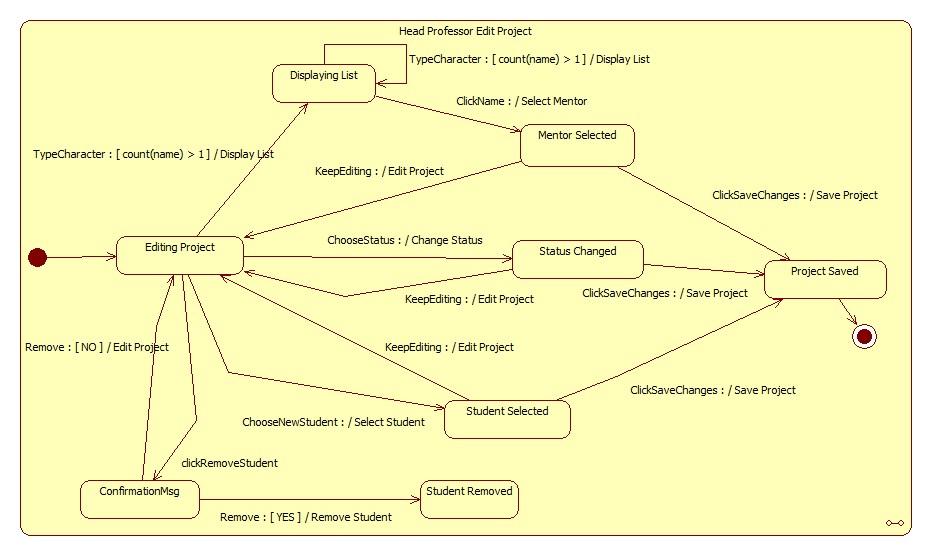
**The following dynamic models are from SPWv.3:**

**Login State Machine**



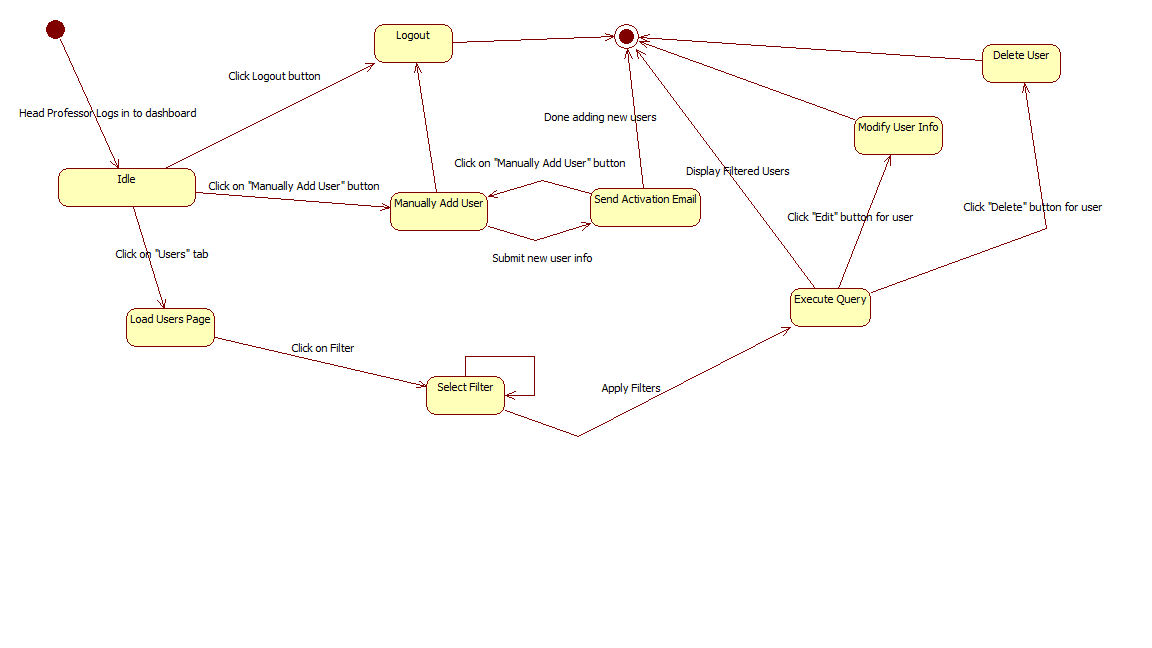
**Profile Subsystem State Machine**

**Head Professor Edit Project**



## Activity Diagram for Repository:

****

**Activity Diagram for User Management: **

Professor Login (SPW2\_103)



Guest Access (SPW2\_102)



Logout (SPW2\_104)





























Upload File (SPW3\_710)



Download File (SPW3\_720)

Delete File (SPW3\_730) 

Add New Milestone (SPW3\_901) 

Edit Milestone (SPW3\_902) 

Delete Milestone (SPW3\_903) 



Update Minimum (SPW3\_210)



* 1. **Code Specification**

Code specification details remain unchanged from SPWv.4. The following is almost entirely from SPWv.4 design document regarding that section.

The SPWv.1 and SPWv.2 were built using the Code Igniter framework, which is a powerful, lightweight PHP framework that enforces the MVC architecture. Therefore, the code of the SPWv.1 and SPWv.2 reflect the implementation of this pattern by having three main subsystems: model, view, and controller. The SPWv.3 will continue to use Code Igniter for its development, thus SPWv.3 will also maintain the MVC architecture. As a result, the code specification from SPWv.1 and SPWv.2 remain unchanged in SPWv.3.

For the Controller subsystem all classes inherit from the CI\_Controller, a class from the Code Igniter framework that handles all URI requests. For the Model subsystem all classes inherit from the CI\_Model, a class from the Code Igniter framework that uses a modified version of the Active Record Database Pattern. This pattern allows information to be retrieved, inserted, and updated in your database with minimal scripting. Therefore by inheriting from this class, all SPW model classes are able to use the Active Record Database Pattern capabilities implemented by the Code Igniter framework. For the View subsystem, all views and sub views are comprised of HTML and CSS code. All main views are inside the View package and use the header and footer views. Besides the main view, the SPW contains views related to each subsystem that will display to the user the appropriate information for that subsystem.

1. Glossary

*Senior Project Website V5*



EULA: End-User License Agreement

FIU: Florida International University

Google Docs: Free web-based office suite offered by Google within Google Drive service.

PHP: Open source server-side scripting language designed for web development to produce dynamic web pages.

SPW: Senior Project Website

SPWv.1: Senior Project Website Version 1

SPWv.2: Senior Project Website Version 2

SPWv.3: Senior Project Website Version 3

SPWv.4: Senior Project Website Version 4

SPWv.5: Senior Project Website Version 5

**NRMP**: A form of matchmaking devised by the national residency matchmaking program (NRMP). The details of it are that hospitals have residency spots to fill and a ranked list of applicants they want filling them. Applicant ranked residency programs themselves. The matchmaking process works by having applicants apply to residencies in their list’s order. If another applicant tries to displace a current tentative applicant they displace the tentative applicant if the challenging applicant is more desired by the program else they try the next program, displaced applicants try their next program too. Matching ends when all applicants are match, or all applicants are match minus ones who went through their entire list.

**Heuristics**: Colloquially means “common sense approach”, in computer science the meaning is adapted to mean a technique applied to solve a problem. I.e. to shorten average job time in a computer do short jobs first.

**VIP**: Very important project, a project ranked by the head professor to be of a score between 2 and 100 this project will undergo intense matchmaking using heuristics to find a perfect team quickly (higher scored means higher priority)

**OP**: Other project, a project ranked by the head professor to have a score of 1, such projects will undergo NRMP matchmaking to give the ability to compromised project proposer and student constraints. These are more hands-off.

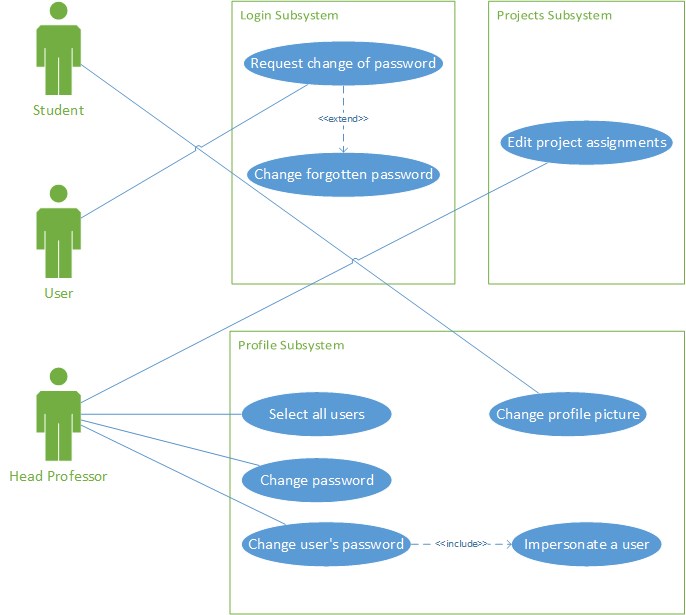
**LinkedIn:** Social media website for business professions to network and advertise themselves

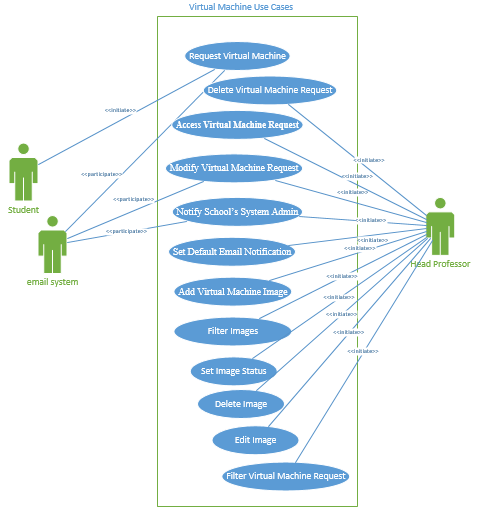
1. Appendix

*Senior Project Website V5*



**5.1 Appendix A Used Case Diagrams (SPWv.5)**





**5.2. Appendix A Used Cases (SPWv.5)**

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_100)** | **Edit project assignments** |
| ***Participating Actor*** | Head Professor |
| ***Flow of Events*** | 1. Head Professor clicks on “Add Students to the Project” drop down menu. 2. System displays a list of active students. 3. Head Professor selects one student from the list and clicks on “Save Changes”. 4. Student is added to the project; system displays added student on the page. 5. At the bottom of the page system displays interest and qualification of selected students. |
| ***Alternative Flows*** | 1. Selected project has not enough room for adding another student, and “Add Students to the Project” drop down menu does not appear. 2. Head professor can sort students in the list by their skills and interest. |
| ***Entry Conditions*** | * Head Professor is logged into the site. * Head Professor is in edit project view (accessible by clicking on a project’s name in “Current Projects” tab). * Project has available space (capacity) for students to be assigned. |
| ***Exit Conditions*** | * Head Professor successfully assigned a student to a project. |
| ***Exceptions*** |  |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_101)** | **Change user’s password** |
| ***Participating Actor*** | Head Professor |
| ***Flow of Events*** | 1. Head Professor clicks on “View All Users” button. 2. System navigates to the user management page. 3. Head Professor selects one of the users and clicks on “Act As User” link. 4. System changes logged in user to the selected user, and navigates to the home page of that user. 5. Head Professor clicks on the user’s profile icon located on the menu bar. 6. System displays user’s profile dropdown menu. 7. Head professor clicks on “Head Professor” option located on the dropdown menu. 8. System navigates to user’s profile page. 9. Head Professor clicks “Click to change password” link. 10. System navigates to change password page. 11. Head Professor inserts new password in the top form, then inserts the same password in the bottom form; finally clicks on “Save Changes”. 12. System redirects Head Professor back to user’s profile page, and displays a message about successful change of password. |
| ***Alternative Flows*** | 1. Head Professor does not enter the same password for the two entries provided on the Reset Password page. In this case, System warns Head Professor and allows Head Professor to retry until the same password is entered for both entries. |
| ***Entry Conditions*** | * Head Professor is logged in. * Head Professor has navigated to admin dashboard. |
| ***Exit Conditions*** | * Head Professor successfully resets his/her account password. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_102)** | **Impersonate a user** |
| ***Participating Actor*** | Head Professor |
| ***Flow of Events*** | 1. Head Professor clicks on the Admin tab on the menu bar, and clicks the “View All Users” button on the dashboard. 2. The system displays a list of all the users in the database. 3. Head professor clicks on “Act As User” located under User’s picture. 4. The system changes logged in user to the selected user, and allows the Head Professor to act on behalf of that user. |
| ***Alternative Flows*** | 1. Head Professor selects filtering options, and then submits a query. This results in the system returning only users that match Head Professor’s search criteria. |
| ***Entry Conditions*** | * Head Professor in logged into the site. |
| ***Exit Conditions*** | * Head Professor successfully poses as selected user. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_103)** | **Change password** |
| ***Participating Actor*** | Head Professor |
| ***Flow of Events*** | 1. Head Professor clicks on the user’s profile icon located on the menu bar. 2. System displays user’s profile dropdown menu. 3. Head professor clicks on “Head Professor” option located on the dropdown menu. 4. System navigates Head Professor to user’s profile page. 5. Head Professor clicks on “Click to change your password”. 6. System navigates to change password page. 7. Head professor inserts current password and new password, and then clicks “Save Changes”. 8. System navigates back to user’s profile page, and displays a message about successfully changed password. |
| ***Alternative Flows*** | 1. Head Professor does not enter correct current password, which causes the system to prompt Head Professor for correct current password. |
| ***Entry Conditions*** | * Head Professor in logged into the site. |
| ***Exit Conditions*** | * Head Professor successfully changed profile password. |

|  |  |
| --- | --- |
| ***Entry Conditions*** | * Head Professor is logged in. * Head Professor has navigated to admin dashboard. |
| ***Exit Conditions*** | * Head Professor successfully changed profile password. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_104)** | **Request change of password** |
| ***Participating Actor*** | User |
| ***Flow of Events*** | 1. User clicks on the “Forgot Password?” link. 2. System displays a dialog prompt, containing a form asking for user’s email address. 3. User enters email address associated with his/her account, and clicks on “Send” button. 4. System verifies that the inserted email address exists in the system, and the user is active. If so, the system sends a message to the given email address with instructions for changing password. |
| ***Alternative Flows*** | 1. User does not enter correct email, which causes the System to display a message informing about incorrect email address. 2. User enters correct email, but the account associated with that email is inactive; this causes the System to display a message about inactive account. |
| ***Entry Conditions*** | * User has navigated to login page. * User has a manually created account. |
| ***Exit Conditions*** | * User successfully requested change of profile password. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_105)** | **Change forgotten password** |
| ***Participating Actor*** | User |
| ***Flow of Events*** | 1. User clicks on the link provided in the message from the System. 2. System navigates to the Reset Password page. 3. User inserts new password in the top form, then inserts the same password in the bottom form; finally clicks on “Save Changes”. 4. System checks that both passwords entered are the same. If so, System changes User’s password and redirects to login page; then displays a message informing about successful change of password. |
| ***Alternative Flows*** | 1. User does not enter the same password for the two entries provided on the Reset Password page. In this case, System warns User and allows User to retry until the same password is entered for both entries. |
| ***Entry Conditions*** | * User successfully used forgot password feature to request password change. * User has received the message from the system on his/her email account. |
| ***Exit Conditions*** | * User successfully changed profile password. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_106)** | **Change profile picture** |
| ***Participating Actor*** | Student |
| ***Flow of Events*** | 1. Student clicks on the user’s profile icon located on the menu bar. 2. The system displays user’s profile dropdown menu. 3. Student clicks on his/her name located on the dropdown menu. 4. The system navigates to user’s profile page. 5. Student clicks on the “Choose File” button. 6. System opens file manager window for the user to choose a file to upload. 7. Student selects image file, and then clicks on “Open” in file manager window. 8. System displays dialog prompt asking if Student wants to replace current picture. 9. Student clicks on “Yes”. 10. System displays a message about successful upload. |
| ***Alternative Flows*** | 1. Student clicks on “No” on the dialog prompt, which causes the system to abort the upload. |
| ***Entry Conditions*** | * Student is logged into the system. * Student already has a picture on his/her profile. |
| ***Exit Conditions*** | * Student successfully changed profile picture. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_107)** | **Select all users** |
| ***Participating Actor*** | Head Professor |
| ***Flow of Events*** | 1. Head Professor clicks on “View All Users” button. 2. System navigates to the user management page. 3. Head Professor clicks on “Select All” checkbox. 4. System checks all checkboxes associated with users. |
| ***Alternative Flows*** |  |
| ***Entry Conditions*** | * Head Professor is logged in. * Head Professor has navigated to admin dashboard. |
| ***Exit Conditions*** | * Head Professor successfully selects all users. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_108)** | **Request Virtual Machine** |
| * ***User Story* #148** | * As a student, I want to request a number of virtual machine resources to the head professor, so I have the opportunity to customize my virtual environment with the exact number of virtual machines, the image of preference, the amount of RAM and the storage amount needed for my project. |
| ***Participating Actor*** | Student, email system |
| ***Entry Conditions*** | * Student has logged into Senior Project Website. * Current date is after deadline to choose a project. * Student is in project\_details2 page. |
| ***Flow of Events*** | 1. Student clicks on Create VM-Request button in project\_details2 page. 2. System navigates to VM – Request page. 3. Student creates a VM request by entering in input fields: the image, the amount of RAM, the storage and the number of VMs. 4. System displays in input fields: the input image names, the amount of RAM, the Storage amount and the number of virtual machines. 5. Student clicks on “Submit” button. 6. System stores virtual machine request in database, and notifies Head Professor via email about request. |
| ***Alternative Flows*** | * {From step 3} Student selects virtual machine settings but does not click on “Submit” button and navigates away to a different page. Then, system considers virtual machine request invalid and destroyed. |
| ***Exit Conditions*** | * Student successfully submitted virtual machine request. |
| ***Exceptions*** | * If current day is under deadline and student attempts to access/hack into VM – Request page, system denies student’s access to VM-Request page and prompts warning message. * If student inputs in fields amount of RAM, storage amount and number of virtual machines a non-numerical value. System prompts error message. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_109)** | **Modify Virtual Machine Request** |
| ***Participating Actor*** | Head professor, email system |
| * ***User Story* #146** | As a head professor, I want to check the virtual machine settings requested by students, so I can change the image name, amount of RAM, storage amount and the number of virtual machines if needed. |
| ***Entry Conditions*** | * Head professor is logged into the system. * Head professor is in vm\_requests2 page. |
| ***Flow of Events*** | 1. Head professor modifies a virtual machine request by selecting from drop down menus: the image name, the status of the request and entering in input fields the amount of memory RAM, the storage amount and the number of VMs. 2. System displays the selected image name, selected amount of RAM, selected storage amount, selected number of VMs and the status. 3. Head professor clicks on “Submit” button. 4. System updates virtual machine settings into database and notifies school’s system admin if the status of the request is approved. |
| ***Alternative Flows*** | * {From step 3} Head professor modifies virtual machine request’s settings but does not click on submit button and navigates into a different page. System considers request invalid and it is destroyed. |
| ***Exit Conditions*** | * Head professor successfully modified virtual machine request. |
| ***Exceptions*** | * If email notification input field is empty, system alerts with an error message. * If an invalid email address is used, system alerts with error message. * If head professor inputs in fields amount of RAM, storage amount and number of virtual machines a non-numerical value. System prompts error message. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_110)** | **Notify School’s System Admin** |
| ***Participating Actor*** | Head professor, email system |
| * ***User Story* #144** | As a head professor, I want to be able to send email to our school’s system admins from the Senior Project Web Site based on the students’ requests for the vms that I have approved, so I don’t need to go into my email account and do it manually. |
| * ***Entry Conditions*** | * Head professor is logged into the system * Head professor is vm\_requests2 page * Head professor has modified virtual machine setting. |
| ***Flow of Events*** | 1. Head professor clicks on drop down menu “Status”. 2. System displays on the drop down menu the available status for the virtual machine request. 3. Head professor selects status equal to “APPROVED”. 4. System displays on the drop down menu the selected status. 5. Head professor enters on the email filed the school’s system admin email address and clicks on “Submit” button. 6. System updates virtual machine status in the database and notifies school’s system admin. |
| ***Alternative Flows*** | * {From step 3} Head professor selects from the drop down menu the “APPROVED” status but does not click on submit button and navigates into a different page. System considers request invalid and it is destroyed. |
| ***Exit Conditions*** | * Head professor successfully notified school’s system admin. |
| ***Exceptions*** | * If email notification input field is empty, system alerts with an error message. * If an invalid email address is used, system alerts with error message. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_111)** | **Access Virtual Machine Request** |
| ***Participating Actor*** | Head professor |
| * ***User Story* #145** | As a head professor, I want to receive an email notification when a student requests for one or more vms, so I am able to easily find the request, using a link included in the email notification. |
| * ***Entry Conditions*** | * Student successfully submitted a virtual machine request and notified head professor via email about it. * Head professor is in his fiu.edu email account and in the student’s email notification. * Head professor is logged into the system. |
| ***Flow of Events*** | 1. The use case begins when the head professor clicks on the existing link on the email’s body page. 2. System navigates to VM – Requests page in Senior Project Website and loads all virtual machine request. |
| ***Alternative Flows*** | * {From step 2} Head professor clicks on the link in the email’s body but he is not logged in the system. Then system prompts him with the login page. |
| ***Exit Conditions*** | * Head professor successfully accessed virtual machine request. |
| ***Exceptions*** | * If a user other than head professor attempts to access/hack into VM – Requests page, system denies user’s access to VM-Requests page and prompts warning message. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_112)** | Set Default Email Notification |
| ***Participating Actor*** | Head professor |
| * ***User Story* #145** | As a head professor I want to be able to choose a default person’s email to notify him about vm creation, so I don’t have to memorize his email, neither input it every time manually. |
| * ***Entry Conditions*** | * Head professor is logged in the system. * Head professor is in admin\_dashboard page. |
| ***Flow of Events*** | 1. Head professors enters school’s system admin name in the input field named “Full Name”. 2. System shows inputted name. 3. Head professor enters school’s system admin name in the input filed named email. 4. System shows inputted email address. 5. Head professor clicks on “Set Default Email” button. 6. System validates email address and process request. 7. Head professor sees default name and email address. |
| ***Alternative Flows*** | * {From step 3} Head professor enters name and default email address of school’s system admin, but does not click on “Set Default Email” button and navigates into a different page. System considers request invalid and it is destroyed. |
| ***Exit Conditions*** | * Head professor successfully set/changed default name and email address. |
| ***Exceptions*** | * If head professor leaves “Full Name” filed empty and clicks on “Set Default Email” button, system shows error message. * If head professor leaves “Email” field empty and clicks on “Set Default Email” empty, system shows error message. * If head professor enters an invalid email address, system shows error message. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_113)** | Add Virtual Machine Image |
| ***Participating Actor*** | Head professor |
| * ***User Story* #218** | As a head professor I want to be able to add new available virtual machine images in the system, so that students can choose the newly added images in their virtual machines requests. |
| * ***Entry Conditions*** | * Head professor is logged in the system. * Head professor is in admin\_dashboard page. |
| ***Flow of Events*** | 1. Head professor clicks on “VM Images” button. 2. System navigates to vm\_images page. 3. Head professor enters the new image in the input field named “Image Name”. 4. System shows inputted image name. 5. Head professor clicks on “Add Image Name” button. 6. System processes request and prompts success message. |
| ***Alternative Flows*** | * {From step 3} Head professor enters image name, but does not click on “Add Image Name” button and navigates into a different page. System consider action invalid and it is destroyed. |
| ***Exit Conditions*** | * Head professor successfully added a new virtual machine image. |
| ***Exceptions*** | * If head professor leaves “Image Name” filed empty and clicks on “Add Image Name” button, system displays an error message. * If head professor enters an image name that already exists on the system, system displays an error message. * If a user other than head professor attempts to access/hack into image name page, system denies user’s access to such a page and prompts warning message. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_114)** | Filter Images |
| ***Participating Actor*** | Head professor |
| * ***User Story* #355** | As a head professor I want to be able to filter the available image’s name in the system, so I have the option to easily and more efficiently search through the existing images in the system. |
| * ***Entry Conditions*** | * Head professor is logged in the system. * Head professor is in vm\_images page. |
| ***Flow of Events*** | 1. This use case starts when the head professor selects from a drop down menu one of the available options “All Images”, “ACTIVE”, “INACTIVE” and/or input the full or partial name of an image. 2. System displays the selected filter option. 3. Head professor hits enter key. 4. System processes request and prompts the images according to the selected filter option. |
| ***Alternative Flows*** | * {From step 1} Head professor enters on the input field the full or partial name of an image and hits the enter key. System processes request and prompts all images that matched the search criteria. |
| ***Exit Conditions*** | * Head professor successfully filtered images. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_115)** | Set Image Status |
| ***Participating Actor*** | Head professor |
| * ***User Story* #355** | As a head professor I want to be able to set the status of the images in the system, so just active images are offered to students. |
| * ***Entry Conditions*** | * Head professor is logged in the system. * Head professor is in vm\_images page. |
| ***Flow of Events*** | 1. This use case starts when the head professor clicks on an existing semaphore link/button in vm\_images page. 2. System collects image’s information, processes change status request and prompts success message. |
| ***Exit Conditions*** | * Head professor successfully changed images status. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_116)** | Delete Image |
| ***Participating Actor*** | Head professor |
| * ***User Story* #355** | As a head professor I want to be able to delete an image from the system, so images that are no longer in used can be removed. |
| * ***Entry Conditions*** | * Head professor is logged in the system. * Head professor is in vm\_images page. |
| ***Flow of Events*** | 1. This use case starts when the head professor clicks on an existing “delete” link/button in vm\_images page. 2. System prompts confirmation message for deletion. 3. Head professor clicks on “okay” button. 4. System collects image’s information, processes delete request and prompts success message. |
| ***Alternative Flows*** | * {From step 3} Head professor clicks on “Cancel” button. System considers request invalid and cancel it. |
| ***Exit Conditions*** | * Head professor successfully deleted an image. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_117)** | Edit Image |
| ***Participating Actor*** | Head professor |
| * ***User Story* #355** | As a head professor I want to be able to edit an image from the system, so I can upgrade existing images from the system or correct the ones that already exist if need be. |
| * ***Entry Conditions*** | * Head professor is logged in the system. * Head professor is in vm\_images page. |
| ***Flow of Events*** | 1. This use case starts when the head professor enters in the input field a new image name. 2. System displays the information on the input field. 3. Head professor clicks on “Submit” button. 4. System collects image’s information, processes request and prompts success message. |
| ***Alternative Flows*** | * {From step 3} Head professor enters image name, but does not click on “submit” button and navigates into a different page. System consider action invalid and it is destroyed. |
| ***Exit Conditions*** | * Head professor successfully edited an image. |
| ***Exceptions*** | * If head professor clicks on “Submit” button and input filed is empty. System alerts with an error message. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_118)** | Filter Virtual Machine Requests |
| ***Participating Actor*** | Head professor |
| * ***User Story* #355** | As a head professor I want to be able to filter all virtual machine requests on the system, so I can easily access the information that is of interest to me. |
| * ***Entry Conditions*** | * Head professor is logged in the system. * Head professor is in vm\_requests2 page. |
| ***Flow of Events*** | 1. This use case starts when the head professor selects one or more of the available options to filter virtual machine requests, by status, amount of memory RAM, storage, number of VMs and/or inputs the full or partial name of a student and a term. 2. System displays the selected filter option. 3. Head professor hits the enter key. 4. System processes request and prompts virtual machine requests according to the selected filter options. |
| ***Alternative Flows*** | * {From step 1} Head professor enters on the input fields the full or partial name of an image, student name, term and hits the enter key. System processes request and prompts all images that matched the search criteria. |
| ***Exit Conditions*** | * Head professor successfully filtered virtual machine requests. |

|  |  |
| --- | --- |
| ***Use Case Name* (SPW5\_119)** | Delete Virtual Machine Request |
| ***Participating Actor*** | Head professor |
| * ***User Story* #412** | As a head professor I want to be able to delete a virtual machine request from the system, so virtual machine requests that are no longer needed can be removed. |
| * ***Entry Conditions*** | * Head professor is logged in the system. * Head professor is in vm\_requests2 page. |
| ***Flow of Events*** | 1. This use case starts when the head professor clicks on an existing “delete” link/button in vm\_requests2 page. 2. System prompts confirmation message for deletion. 3. Head professor clicks on “okay” button. 4. System collects virtual machine request’s information, processes delete request and prompts success message. |
| ***Alternative Flows*** | * {From step 3} Head professor clicks on “Cancel” button. System considers request invalid and cancel it. |
| ***Exit Conditions*** | * Head professor successfully deleted a virtual machine request. |

**5.3 Appendix C Class Interfaces (SPWv.5)**

**Project Controller**:

Declaration: class ProjectController extends CI\_Controller

public function \_\_construct()

{

parent::\_\_construct();

$this->load->helper('project\_summary\_view\_model');

$this->load->helper('request');

$this->load->helper('flash\_message');

load\_project\_summary\_models($this);

$this->load->model('SPW\_Project\_Details\_View\_Model');

$this->load->model('spw\_notification\_model');

$this->load->model('spw\_vm\_request\_model');

$this->load->model('spw\_user\_model');

}

Methods:

public function deleteVMRequest($request\_id,$image,$f\_ram,$storage,$f\_qty,$status,$name,$term)

public function filterVMRequests($image,$f\_ram,$storage,$f\_qty,$status,$name,$term)

public function vm\_requests()

public function loadEditImage()

public function editImage()

public function deleteImage($delete\_image\_name, $image, $status)

public function changeImageStatus($image\_name, $change\_status, $image, $status)

public function filterImages($image, $status)

public function vm\_images()

public function addImages()

public function vm\_request()

**Virtual Machine Model**

*Declaration*: class SPW\_vm\_request\_Model extends CI\_Model

*Constructor:*

public function \_\_construct()

{

parent::\_\_construct();

}

*Methods:*

public function searchFilteredVms($where)

public function getVMRequests()

public function getUserRequests($user\_id)

public function deleteVMRequest($request\_id)

public function insertVmRequests($requests,$user\_id)

public function updateRequestsFromProject($requests)

public function getHeadEmail()

public function getProjectTitle($project\_id)

public function searchFilteredImages($where)

public function getVMDefaultEmailCreation()

public function getAllImages()

public function addImage($image)

public function updateImageStatus($status,$image)

public function deleteImage($image)

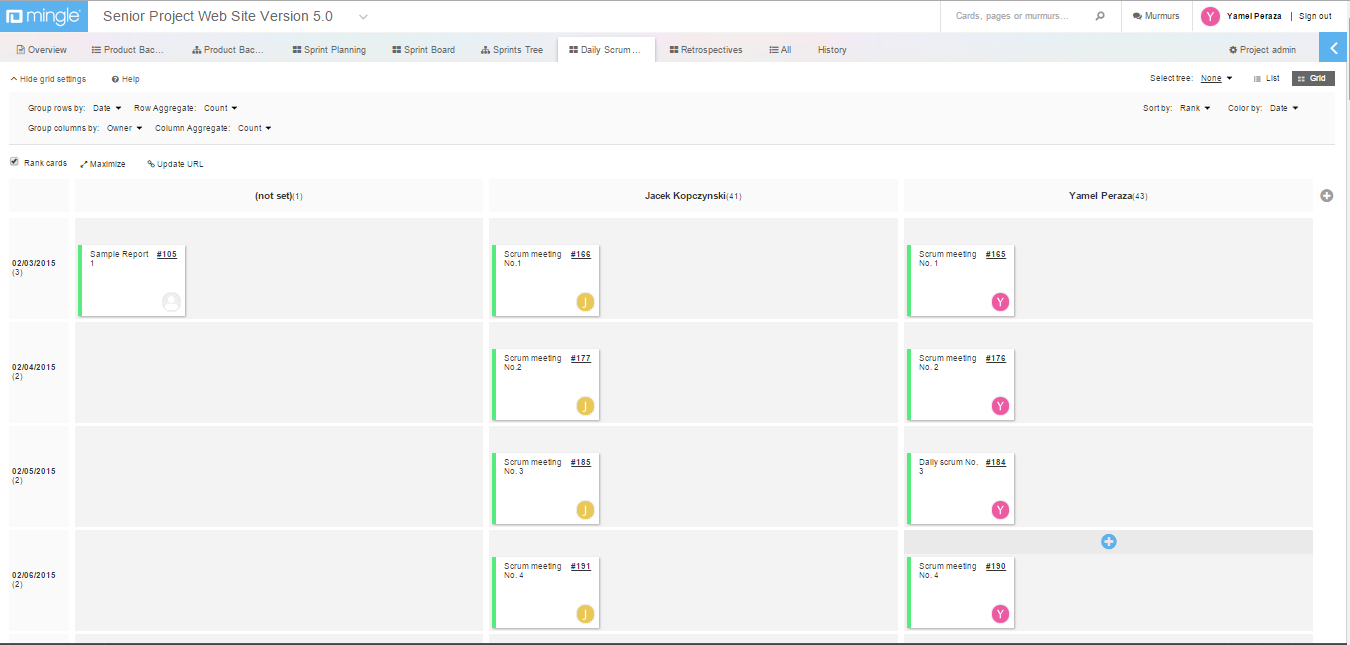
public function editImage($old\_image, $new\_image)

public function updateImageRequests($requests)

public function getActiveImages()

**5.4 Appendix D Diary of Meetings and Tasks**

Developer members of the team met on a daily basis. Below is a screenshot of our daily meeting using Mingle. Meeting with product owner and lead developer occurred weekly resulting in the proposed implementation generally described in this document.



1. References

*Senior Project Website V4*



Julio Perez "*Design Document for Senior Project Website: Version 4*" (Summer 2014). Florida International University.

Christopher Kerrutt, William Marquez, Cynthia Tope. "*Design Document for Senior Project Website: Version 3*" (Spring 2014). Florida International University.

National Resident Matching Program “The Match” Retrieved January 25th 2014, from http://www.nrmp.org/match-process/match-algorithm/

Arbelaez, A., Hamadi, Y., & Sebag, M. (n.d.). Online Heuristic Selection in Constraint Programming. Retrieved July 07, 2014, from research microsoft: http://research.microsoft.com/en-us/people/youssefh/search-socs.pdf