3. Architectural Design

In this section a high-level description of the system and its modules is presented.

3.1. Architecture Diagram

In this section the design of the system will be described using the 4+1 Architectural View Model (AVM). This model describe the architecture of a system based on multiple complementary views, considering the viewpoints of different stakeholders of the system (e.g. developer, architect, user, etc.). This view model propose to have a physical, logical, development, and process view, supported by different UML diagrams, plus selected use cases.

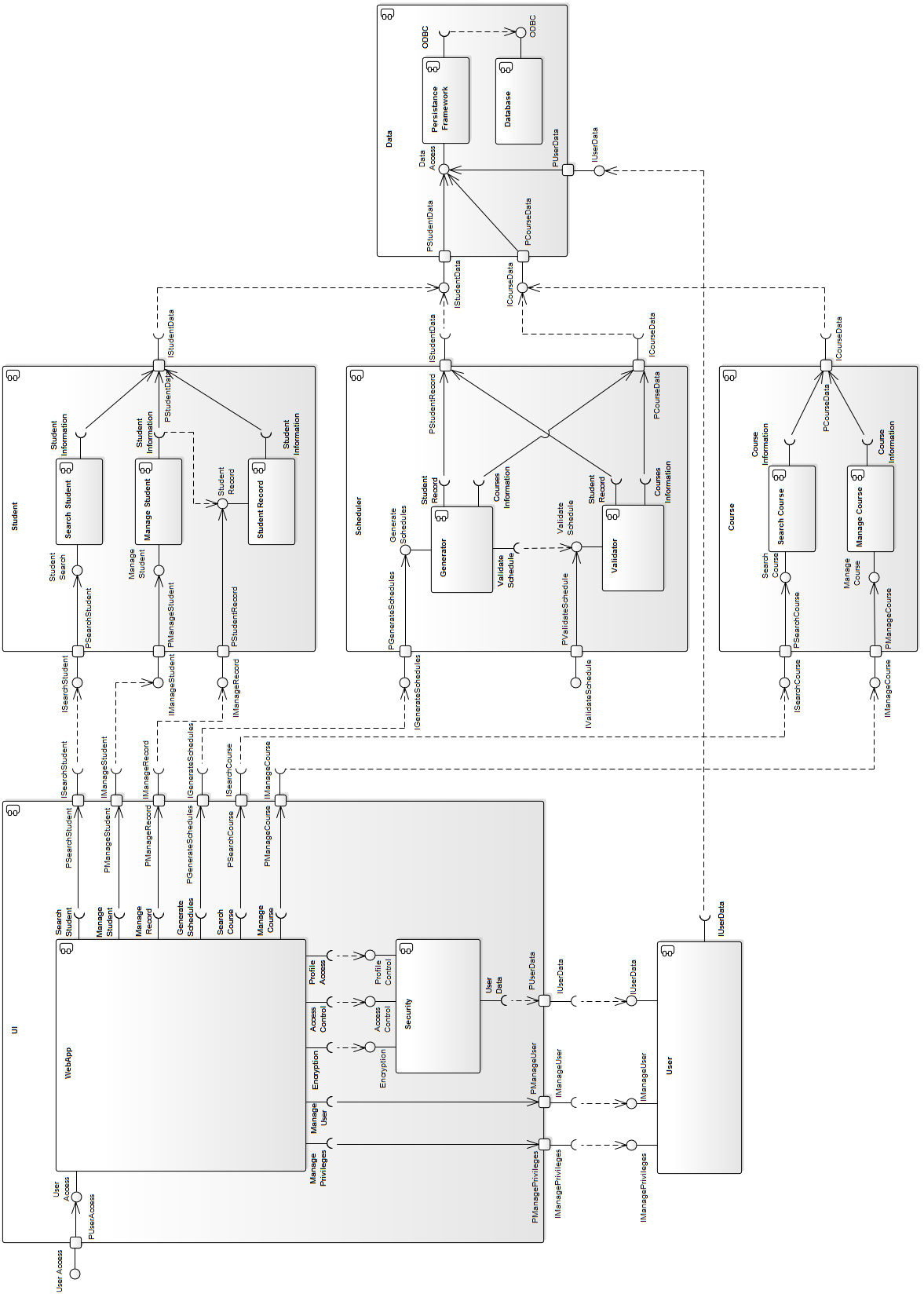
3.1.1. Logical View

In the 4+1 AVM, the Logical View describes the functional aspect of the system. In other words, what the system should provide in terms of service to its users. To represent this view the several UML Diagram can be used (e.g. class, object, sequence, communication, state, etc.) Following the structure of this document (sequence diagrams are shown later), only the Class Diagram is used here, as shown the image bellow.

FullClassDIagram.png

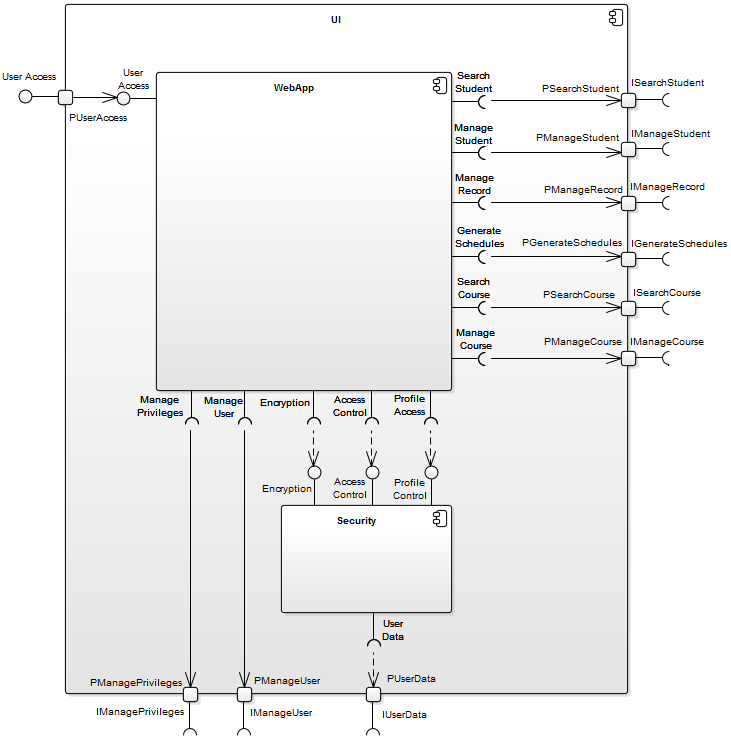
3.1.2. Development View

In the 4+1 AVM, the Development View describes the structural aspect of the system. In it modules and subsystems can be represented using Package and Component diagrams, making it the developer’s view of the system. In the following sections the UML Component Diagram notation has been used to model the component of the system, as shown the image below. As it can be seen all the interfaces dependency are properly depicted (except for the validation schedule interface, which is required only in the scheduler component)



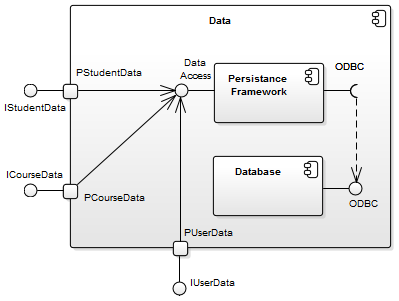
3.1.2.1. Component Diagram “UI”

The UI Component is composed by the components WebApp and Security. The main purpose of the UI component is rendering the web user interface, delegating all the user’s requirements to the corresponding components (i.e. it only has required interfaces). In the other hand, the Security component is continuously invoked to encrypt the information, and validate the access and profile privileges. The Security component provide three interfaces for encryption, access control and profile control. The UI Component represent the view and controller of the Apollo’s MVC architecture.



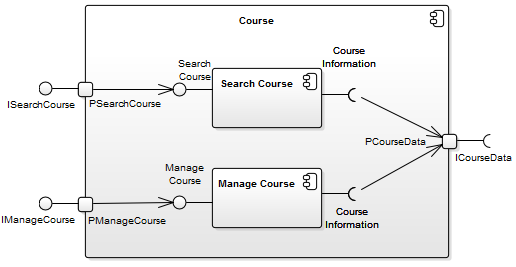
3.1.2.2. Component Diagram “Data”

The main purpose of the Data Component is to provide persistence features to the system’s data. It is composed by the persistence framework used in the system and a database. Although it provide three interfaces, all of them are mapped to one interface to access the system’s date. This component is part of the Model in the Apollo’s MVC architecture.



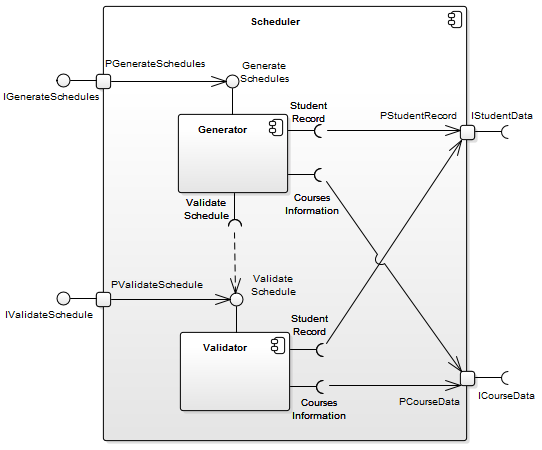
3.1.2.3. Component Diagram “Course”

The Course Component implements the course’s search and manage features, which are at the same time the two components that conform it. These features are available through the two provided interface search course and manage course.



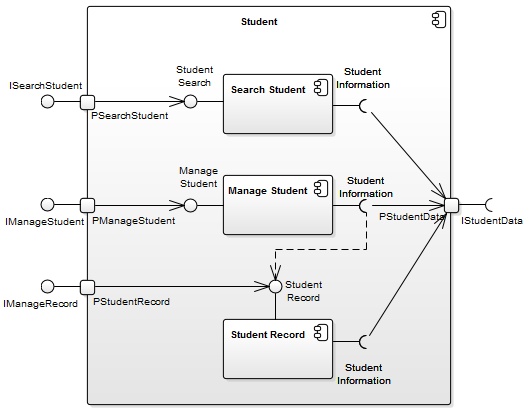
3.1.2.4. Component Diagram “Scheduler”

The Scheduler Component is responsible for the generation and validation of the schedules. To do so it provides two interfaces, but only one is used. To generate and validate the schedules it is required to access information of both the student and courses, so two interfaces are required to.



3.1.2.5. Component Diagram “Student”

The Student component implement the student search and manage features, as well its records. To do so it provide three interfaces, and only one is required in order to access the student information.

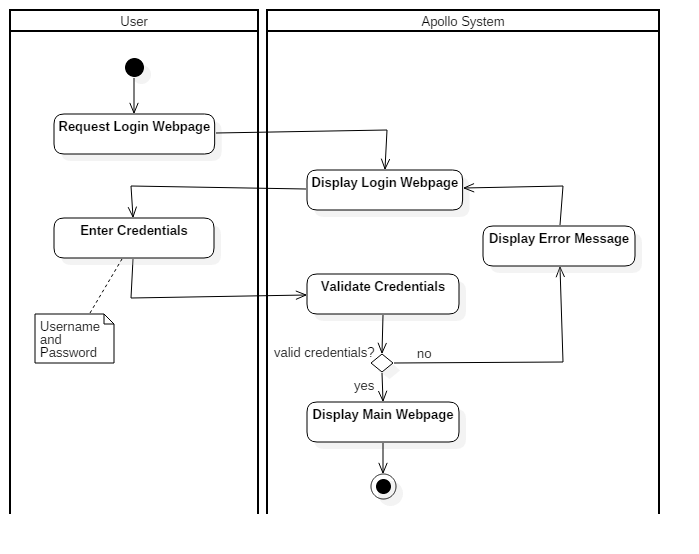


3.1.3. Process View

In the 4+1 AVM, the Process View describes the dynamic aspect of the system. Although it was proposed mainly to represent dynamic behaviour of the system (runtime aspects), it can be used as well for interactions with the users. When referring to software elements, typical concerns about the process view are communication between objects and components, integration, concurrency issues and transactions. When referring to interaction between users and the system, it concerns about functional processes. In this design document, the latter has been addressed. In the following sections the most representative process for each user has been described using UML Activity Diagram notation.

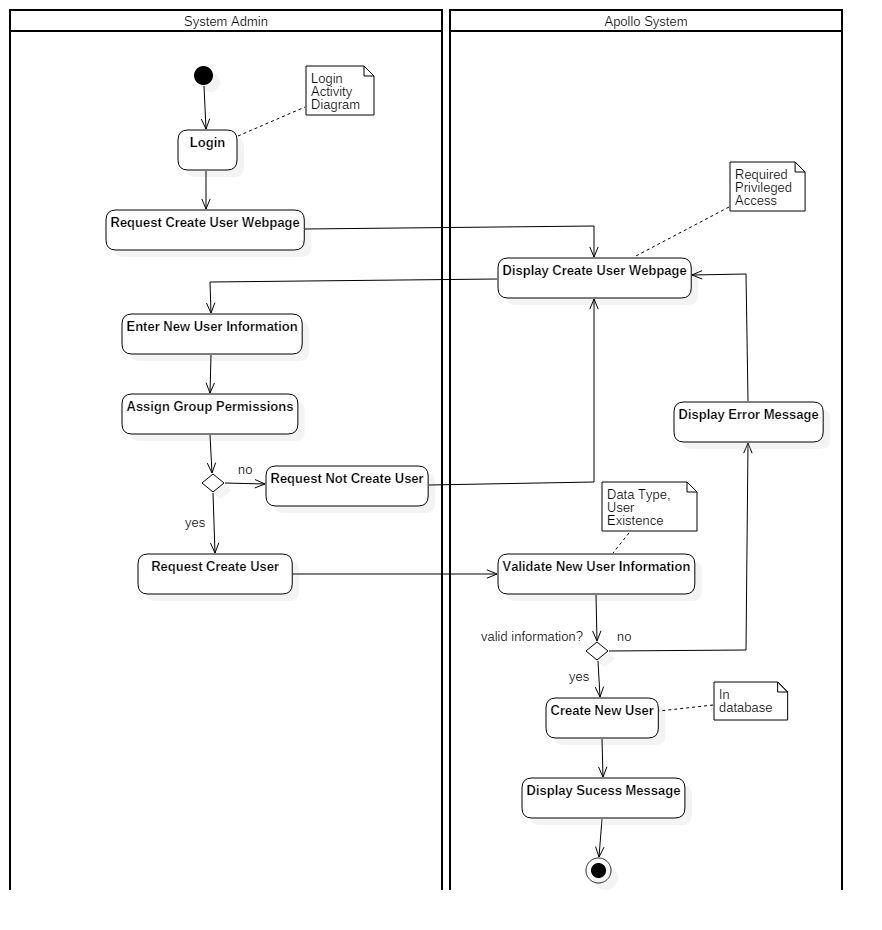
3.1.3.1. Activity Diagram “Login” (by All Users)

The activity diagram shown below depicts the actions performed by any user that attempts to login into the Apollo System. Since the use of the system requires privileged access, prior to displaying any option, the system requires the user to provide correct credentials (i.e. username and password). Once the access has been granted, the system displays profile information (i.e. personal data and allowed actions) associated with the user logged in.



3.1.3.2. Activity Diagram “Create User” (by System Admin)

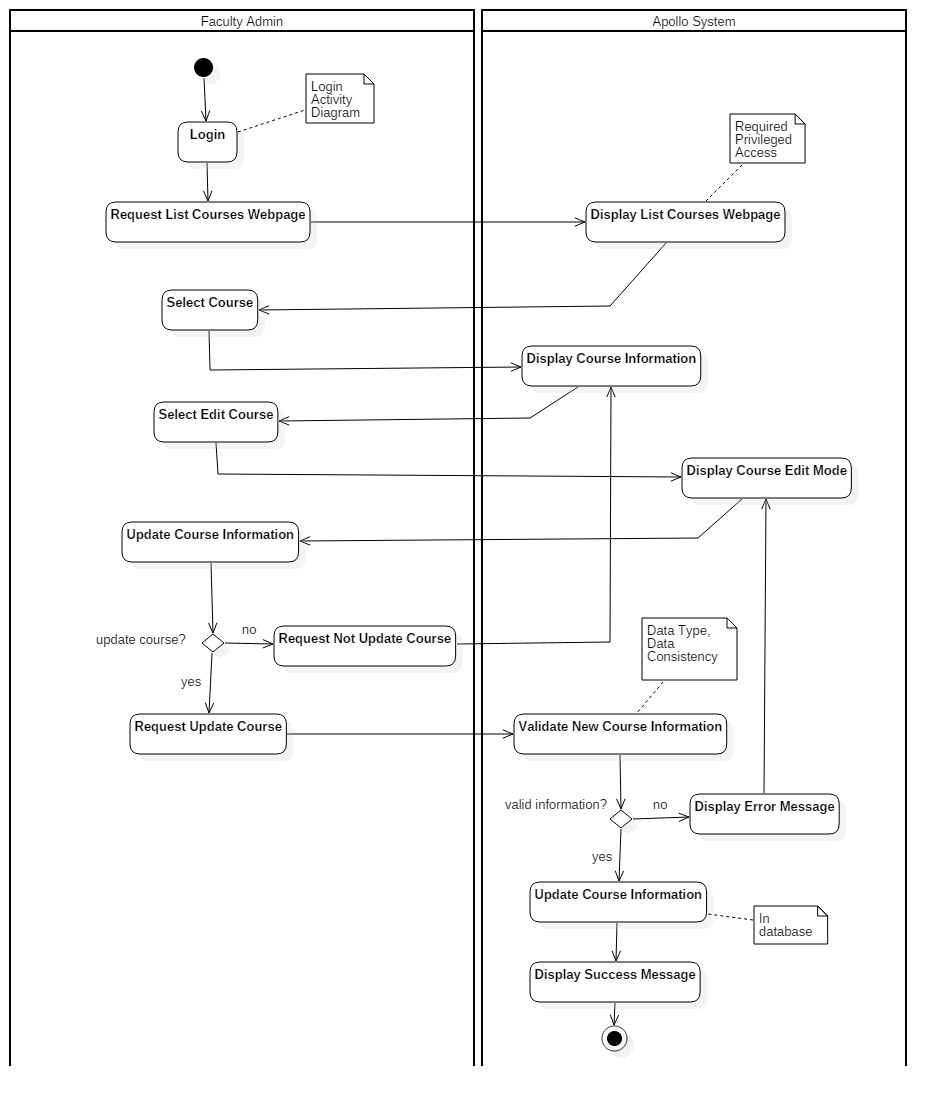
The activity diagram shown below describes the process by which the System Admin creates a user in the Apollo System. Like all the available options, it starts with the login of the user into the Apollo System, process previously described. Once the access has been granted, among all the available options, the user request to display the Create User web page, option only available for users with System Admin privileges, as shown in the diagram. When the web page has been displayed, the new user information is entered along the group permissions. At this point the System Admin can either proceed with the creation or cancel it. If proceed is selected, some standard verification is performed by the Apollo System (like user uniqueness). If the validations are successful, the system creates the new user into the database, and that action is informed to the System Admin. It can be said that this is a standard user creation process, conveniently adopted in the system (by using standard process some non functional requirements are improved, like maintainability).



3.1.3.3. Activity Diagram “Update Course” (by Faculty Admin)

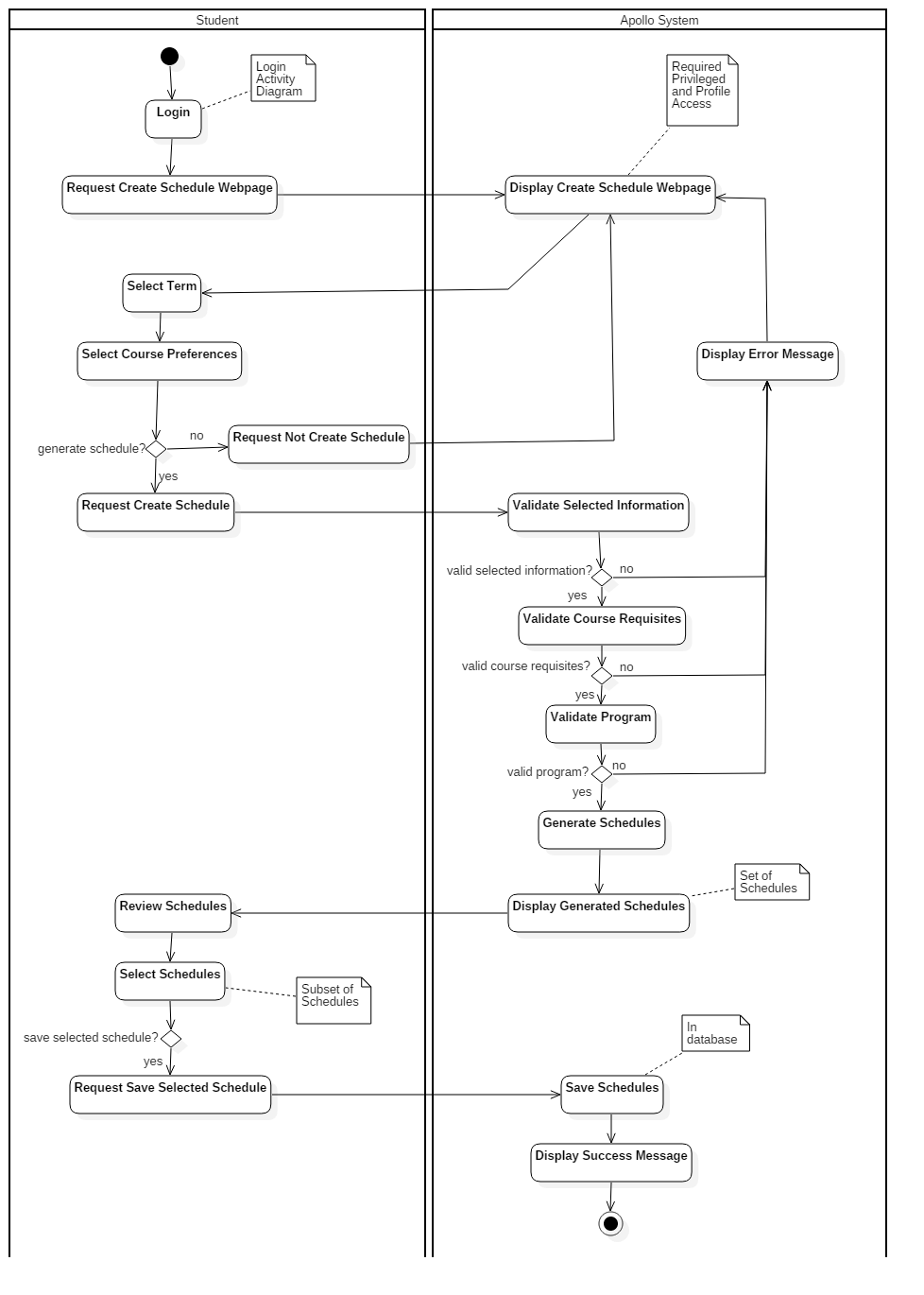
The diagram shown below details the activities performed by a Faculty Admin user in order to update a course’s information. After successful login, the user selects from the menu the option to list all the available courses in the system. From the list displayed, the Faculty Admin selects the course of interest to be reviewed in detail. In that page, the user can select to update it. When this action is requested, the system displays the course in edition mode, which means that all the editable information is available for changes. When the Faculty Admin concludes the update actions, it can select to save the change or discard them. If update the course is selected, some standard validations are conducted before save the information into the database. When the changes are saved, a success message is displayed to confirm the action.

To update a course another process can be followed, when instead of listing all available courses the user can select to search for a specific one. Once the course has been found and opened, the process continues exactly as the one described here (from Select Course).



3.1.3.4. Activity Diagram “Generate Schedules” (by Student)

The last representative activity diagram shown below describes the process followed by a Student to generate his or her schedules. Like all the previous processes, after a successful login, the request to the system displays the web page to generate the schedules. In that page, students can select their preferences to create a schedule (i.e. time, term and courses). After that, he/she can decide either to continue the process or cancel it. When continue is selected, thus create the schedule, Apollo System performs first a serie of required validations (e.g. course grade and requisites) before generating the different schedules. After the generation is completed and displayed, the Student can review and select his/her prefered schedules, requesting to the Apollo System to save them for further review.



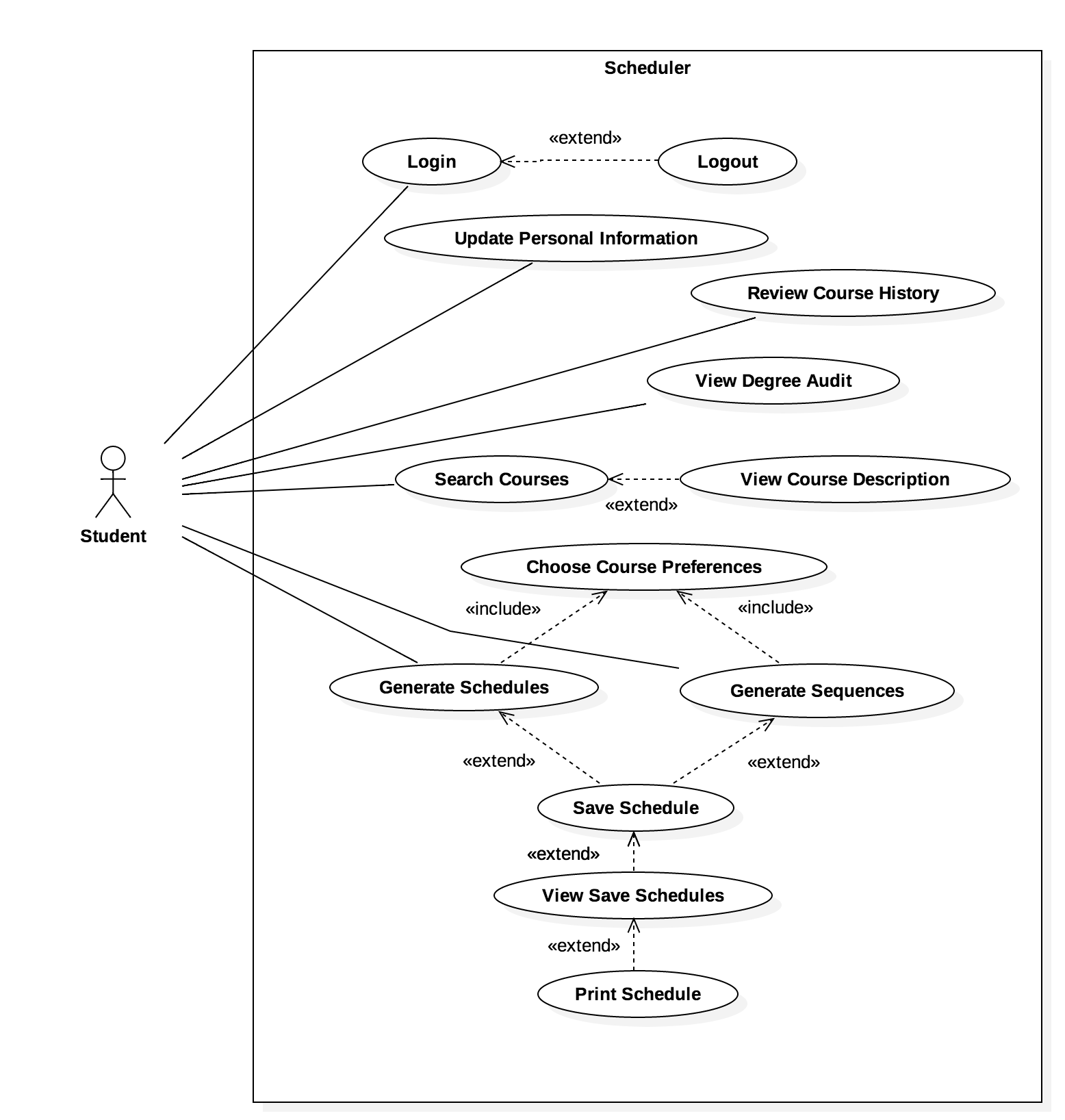
3.1.4. Physical View

deploymentDiagram.png

The deployment of our application is divided into two main physical nodes, as shown in the figure. The client node can be any device with any operating system, as the application is platform-independent. Clients interact with the application through a browser of their choice. The application server is based on the Apache web server and the MySQL database.

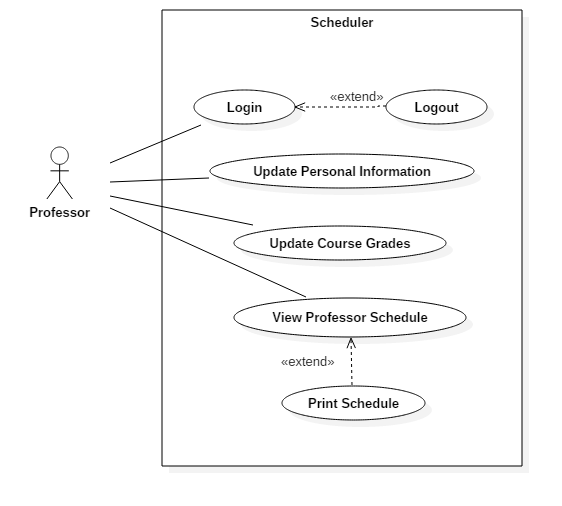
The client communicates with the web server through HTTP and HTTPS. Client requests are received by the Apache server which then interacts with the database schema and tables, and sends responses back to the clients. The Laravel PHP Framework is used to handle manipulations of the database, by using a RESTful approach of controlling resources.

3.1.5. Scenarios

3.1.5.1. Student Scenarios

The student constitutes a user of the system who can interact with the scheduler application in all the common scenarios shown in the diagram. The student has to log in to the application and then potentially log out of it.

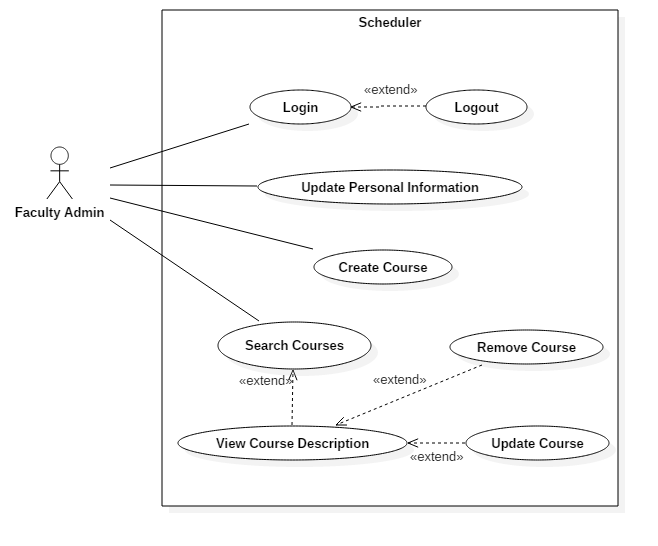
Once logged in, students can update their personal information, review their course history and degree audit, search for courses, and generate schedules (or entire sequences of schedules) based on their preferences. The students can also save the generated schedules, view and print them.

3.1.5.2. Professor Scenarios

The professor must also log in to use the Apollo system, and can later log out. Professors can update their personal information, update grades in a course they instruct, and view and print their personal schedules.

Note: scoped out since it is not necessary for overall functionality of the system.

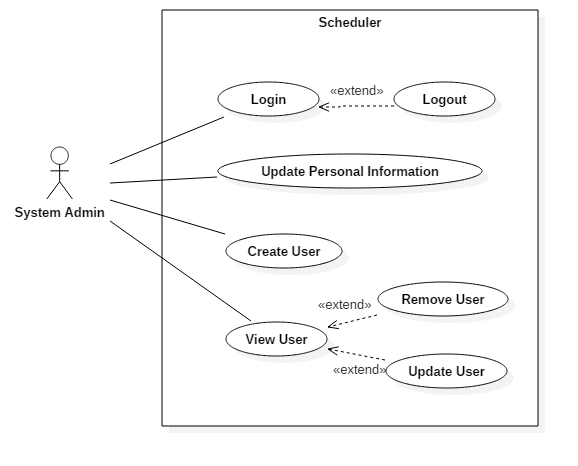
3.1.5.3. Faculty Administrator Scenarios



The Faculty Admin user logs into the system and can later log out of it. This user can update his or her personal information, create courses, search for courses and view, update, or remove the courses found.

Note: scoped out due to time constraints.

3.1.5.4. System Administrator Scenarios



The System Admin, as any other user, has to log in to use the Apollo System and can log out. These administrators can update their personal information, create new users, view existing users and update or remove them.

Note: scoped out due to time constraints.

3.2. Subsystem Interfaces Specifications

*Instructions: [Specification of the software interfaces between the components, i.e. specific messages (or function calls) that are exchanged. These are also often called “Module Interface Specifications”. Description of the parameters passed in these function calls in order to have a service fulfilled. Include valid and invalid ranges of values. Each subsystem interface must be presented in a separate subsection.]*

The following subsystem interface specifications provide an overview of the messages exchanged between components. Each interface specification describes the function calls, parameters used, and rang of invalid and valid values. Please assume that accessor, mutator, and constructor methods are included.

Student Subsystem

ISearchStudent:

This has been scoped out from implementation. This allows professors and faculty admin to search for a student and input grades.

IManageStudent:

This allows user to edit student information such as user settings and preferences.

|  |
| --- |
| **Class Student** |
| Description: Student class provides information about the student, including the course record, the name, the preferences and the password |
| Attributes:   * student\_id: int * first\_name: String * last\_name: String * user\_name: String * password: String * studentRecord: CourseRecord List * Preferences: Preferences |
| Methods:   * getID(): int   Returns an integer of the student’s identification   * getFullName: String   Returns a String of the full name’s student   * changePassword(oldPass: String, newPass)   Changes an old password to a new one by passing oldPass and newPass in the parameters and updating the password with newPass |

|  |
| --- |
| **Class Preferences** |
| Description: Preferences provides the student the ability to change his or her schedule preferences |
| Attributes:   * id: int * day\_of\_week\_id: int * start: time * end: time |
| Methods:   * addPreference(day: int, start: time, end: time)   Adds a preference, by providing the day, start time and end time in the parameters   * editPreference(day: int, start: time, end: time)   Edits an already available preference by proving the day, start time and end time in the parameters |

IManageRecord:

This has been scoped out of implementation. This allows for faculty admin to edit a student academic record.

|  |
| --- |
| **Class CourseRecord** |
| Description: CourseRecord contains all courses taken by the student and the final grade they got |
| Attributes:   * Course: Course * Grade: Grade |
| Methods:   * N/A |

|  |
| --- |
| **Class Grade** |
| Description: Grade manages the grades received by the student for all courses taken |
| Attributes:   * Id: int * Value: String |
| Methods:   * N/A |

Course Subsystem

ISearchCourse

This allows users to search for courses being offered. This functionality is handled by the AngularJS framework.

Schedule Course

IManageCourse

This has been scoped out from implementation. This allows system admins and faculty admins to adjust course information.

Course

Faculty

TimeSlot