Concordia University
Department of Computer Science
and Software Engineering

Software Process
SOEN 341/4 --- Winter 2009 --- Section S

SOFTWARE DESIGN

Team members information	
Name	**SID**
Marc-André Moreau	9347100
Mathieu Dumais-Savard	6095275
Julia Lemire	9402969
Phuong-Anh-Vu Lai	5644399
Sébastien Parent-Charette	9178821
Andreas Eminidis	9377603
Corey Clayton	9349200
Eric Chan	9365079

TABLE OF CONTENTS

TABLE OF FIGURES

1 Introduction

This document examines the 3-tier architecture of the web based application scheduling software designed and implemented by team7. This includes a 4+1 architectural view model of this architecture as well as a break-down of two its main use cases. It includes a description of the different modules of the user interface, a mock-up of the logic to be used and a detailed break-down of the database.

2 Tier Architecture

2.1 Overview

The team7 scheduling system, hereby referred to as PlanZilla, uses a 3-tier software architecture. This means that the presentation, the logic and the data are separated. This has the advantage of making the code more reusable and maintainable since the tiers can be modified without necessarily affecting each other. Figure 1 shows the different components of a 3-tier architecture:

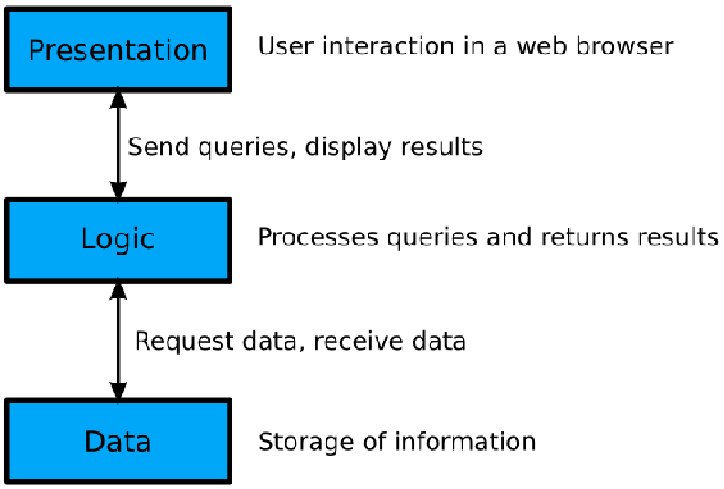

Figure 1 - 3-tier architecture.

In the case of PlanZilla, the presentation is what is presented to the user in the form of a website. This part is done using combination of HTML documents with JavaScript and a library named jQuery to abstract access to the logic tier using AJAX methods and provide us with a a set of rich re-usable user interface controls.

The logic tier is composed of multiple PHP modules for the various tasks the application needs to handle. This includes user authentication, scheduling, registration and courses.

The third tier is the database. The database used in this project is MySQL. The multiple PHP modules interact with the database in order to query data requested by a user in the presentation tier, and return the results in a format ready to be used by the presentation tier.

3 Development View

3.1 Component Diagram

Figure 2 shows a component diagram depicting the 3-tier architecture used for PlanZilla.

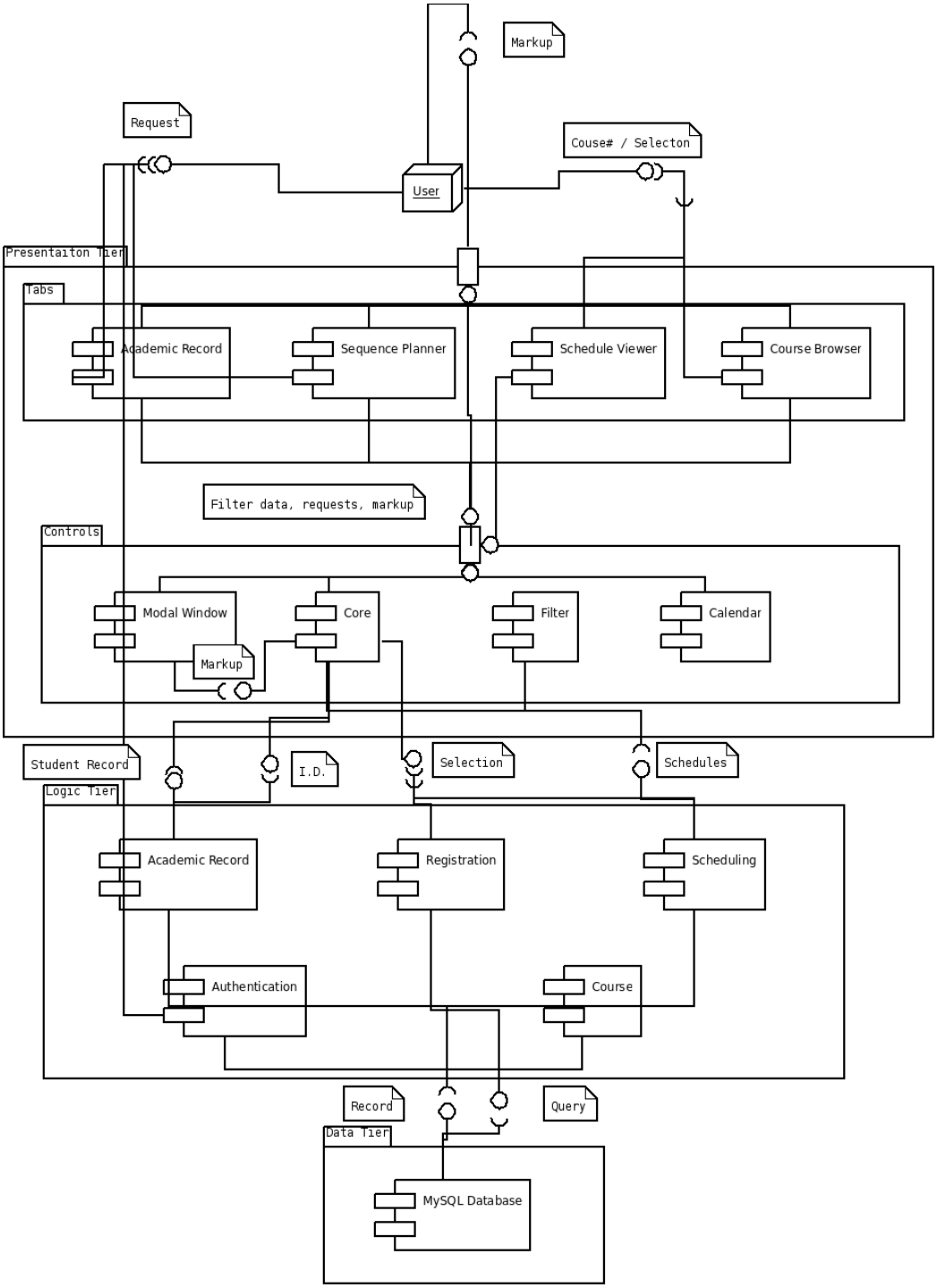

Figure 2 - UML Component Diagram.

3.2 Brief Component Diagram Description

Figure 2 illustrates how the components of Planzilla are connected to each other.

3.2.1 User

Starting at the top of the diagram, the user provides either a page request – i.e. simple tab navigation – or specific data, which may include a course's information or a course selection. The user then receives a markup containing either a JSON string, HTML code or a JavaScript post that is rendered on the page.

3.2.2 Tabs

The tabs package acts as the main part of the user interface. They provide the user with aforementioned markup. At the same time, they provide and receive filtered data, requests and a markup from the core. This markup is likely to contain sectional HTML data.

3.2.3 Controls

The controls package of the presentation tier is mainly composed of the core which acts as a bridge between the tabs component and the logic tier. The component diagram has been modified so that any component from the presentation tier trying to access a component in the logic tier must do so through the core component. The only component that acts as the exception to this is the authentication component. This is because it must receive login credentials directly from the user.

The next three sections describe the modules of each tier in further detail.

4 Presentation Tier

The presentation tier contains all of the logic for the user interface. It is primarily handled by JavaScript functions.

4.1 Tabs

4.1.1 Academic Record

The application has a tab called Academic Records, in which students can view their transcripts and other useful academia related information. The Academic Record module in the presentation tier only handles the presentation of the data that is provided by the analogous Academic Record from the second tier, while this information stored in the MySQL database.

4.1.2 Sequence Planner

The Sequence Planner is a handy tool that can generate potential schedules from a list of given courses that a student wants to register for. It has the ability to propose courses from the student's course sequence. The user also has the option of setting special constraints, such as not having classes late on Friday night or early Monday morning.

4.1.3 Schedule Viewer

The schedule viewer shows a student's schedule in a calendar view. A schedule is only available if the student has registered for courses.

4.1.4 Course List Browser

The course list browser can be accessed by both an authenticated and an unauthenticated user. It provides a full list of the courses offered with some general information. This can be useful for both registered students, who want to learn more about the courses they have to take, and for people currently not enrolled in any program but would like to know more about a give course.

4.2 Controls

4.2.1 Core

The core is the main page of the website. It contains the tab controller. It provides access to the authenticated and the unauthenticated sections with the use of an authentication module. Other formatting features like headers, footers and the main menu are managed in this module. The core is responsible for consuming services provided by the Logic Tier and handle error such as communication issues if any.

4.2.2 Modal Window

At different places in the application a modal window is needed, such as for the authentication of a user. The modal window prompt user for a question and require immediate attention for the process to go further. When a Modal Window is opened, nothing else can be clicked in the application. Instead of rewriting the same code multiple times, a modal window module is used. A sample of a modal window used in the PlanZilla application can be seen in Figure 3.

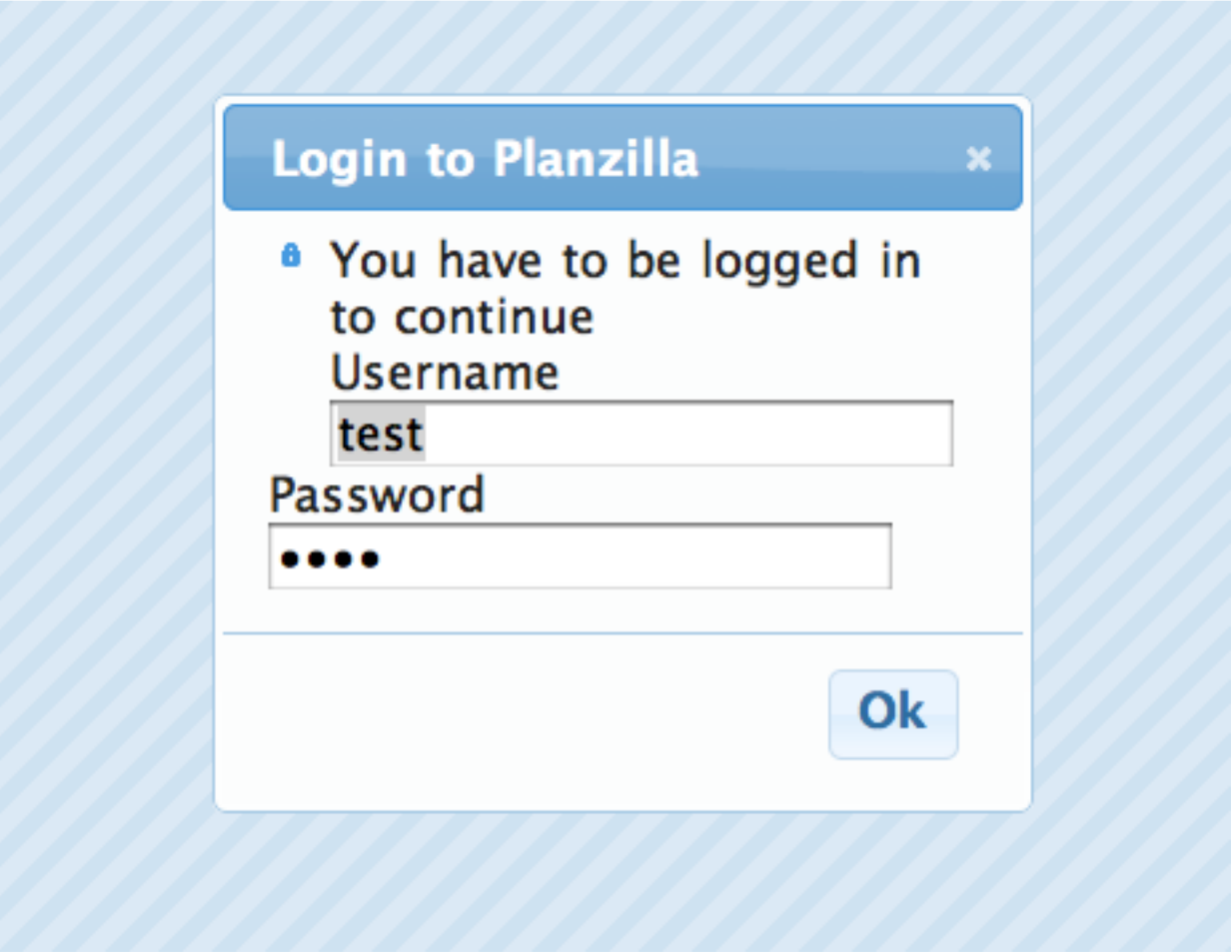

Figure 3 - Example of a modal window.

4.2.3 Filter

The filter is a control used for the auto completion of a text entry in various modules of the presentation tier. For instance, if a user starts typing “Software” the filter could suggest the auto completion “Software Engineering”.

4.2.4 Calendar

The calendar module is used for the presentation of schedules in a sleek calendar view. Both the schedule viewer and the sequence planner access this module. A sample of the calendar can be seen in Figure 4.

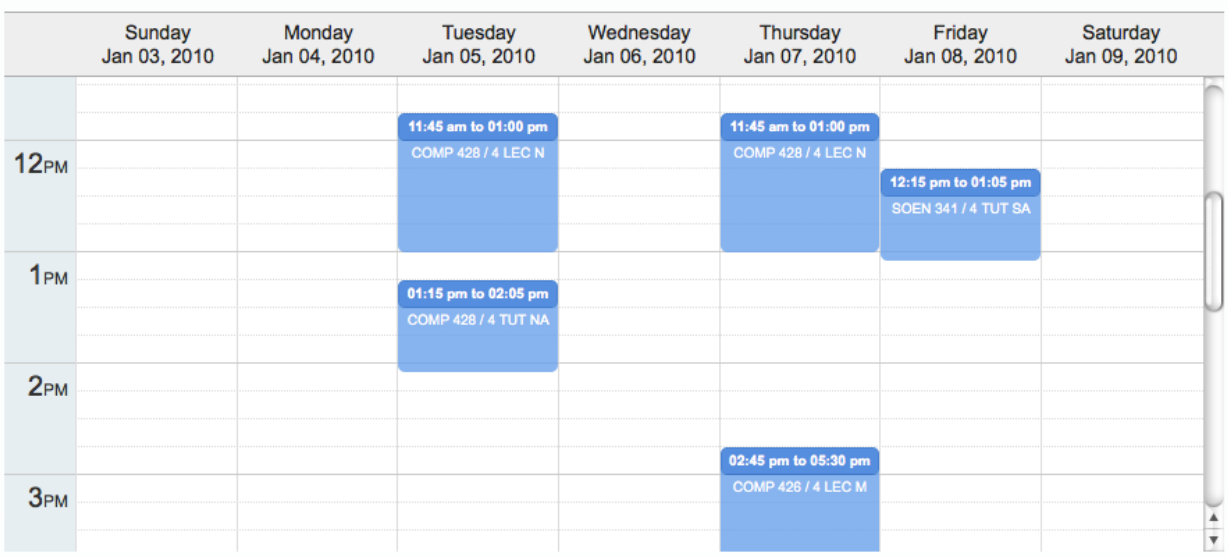

Figure 4 - Example of the calendar.

5 Logic Tier

5.1 Authentication

The Authentication module is responsible for properly validating a user for the duration of their session on the website. It receives a user's credentials and validates them according to the information stored in the database. Passwords are stored with the following hashing function: sha256(sha256(password) + random_salt).

5.2 Academic Record

The Academic Record module in the logic tier has the task of gathering information, such as the grades of a student, and returning this data in the form of a full student transcript. This is to be readily displayed by the Academic Record module from the presentation tier.

5.3 Registration

The Registration module is used when a student has chosen a schedule and now wants to register the selected courses. Registration first uses the Course module to verify that the schedule is valid and that the student can take all the courses selected. The student is then registered in these courses. In the case of an invalid schedule, such as when a course becomes full right before the student wants to register, then the Registration module will deny the request and a brief reason for the error will be given.

5.4 Scheduling

The Scheduling module takes a list of courses and a list of constraints, and generates all of the possible schedules resulting from them. The list of possible schedules is then sent back to the calling module, which in this case is the Sequence Planner module. Scheduling also depends on the Course module in order to verify the validity of the generated schedules. This is important to ensure that a student satisfies all prerequisites of a given course.

5.5 Course

The Course module is used to access course information. It also to handle the logic of validating a course schedule with regards to course dependencies and the courses a student has already taken.

6 Data Tier

6.1 Overview

Figure 5 shows an overview of the database architecture for PlanZilla:

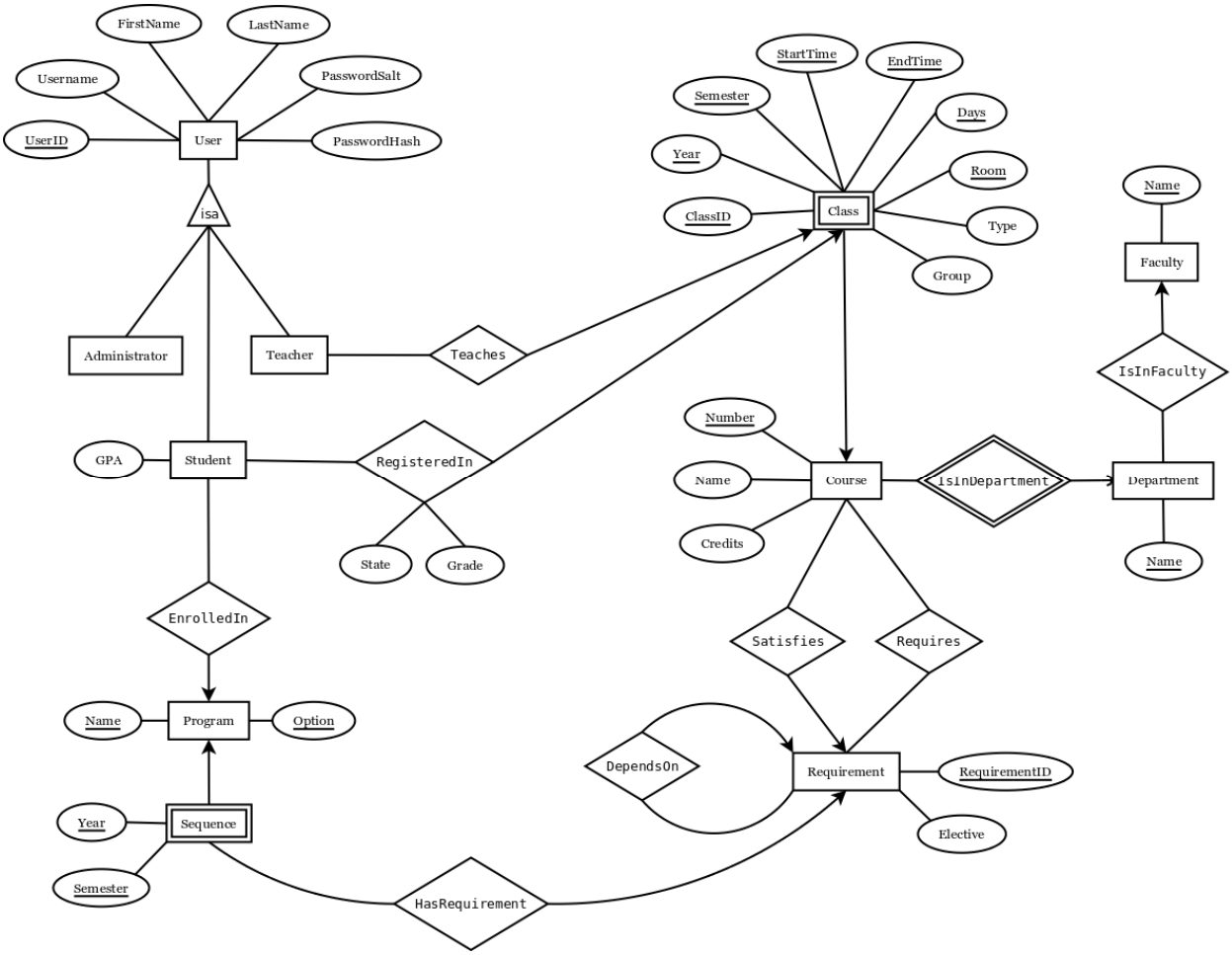

Figure 5 - Entity-Relationship Diagram.

Each part of the above entity-relationship diagram is explained more thoroughly in the following sections.

6.2 Users

Figure 6 shows the different types of users that can access the system.

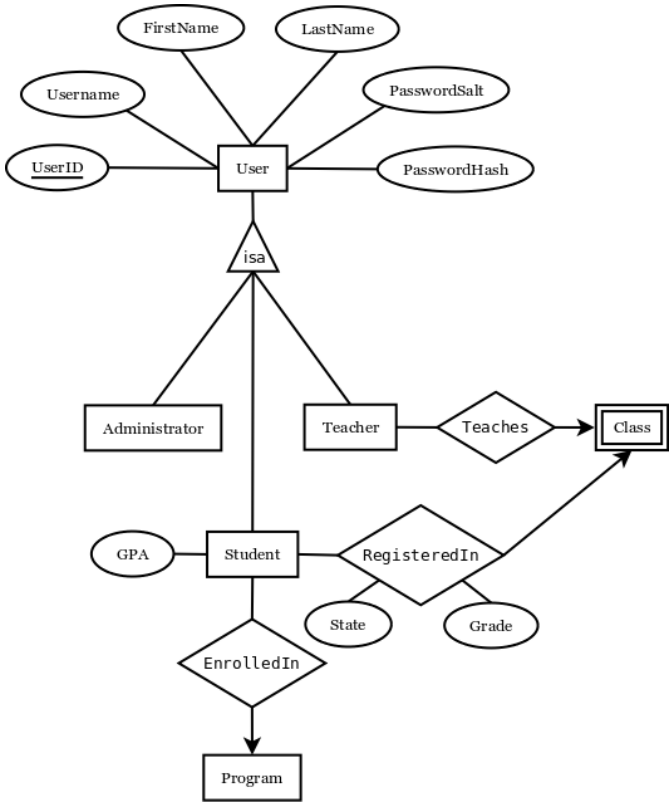

Figure 6 - The different types of users that can access PlanZilla.

6.2.1 Administrator

Users in the administrators table have administrative powers in the database. This would usually be reserved for the staff of the school that have access to operations that other types of users do not. This may include adding new courses and manually registering students for courses that are already full. But such functions are not in the scope of this project.

6.2.2 Student

Students are a special type of users that can be enrolled in a program and registered in courses. It inherits all of the attributes from User, and has an extra attribute for the student's GPA. The RegisteredIn relationship is used for both present and past courses: if a student has successfully passed a course, then the State attribute should say "Pass" and the Grade attribute should contain the associated grade. If the course is currently being taken, then State should say "Current" and no grade should be available yet.

6.2.3 Teacher

Teachers, like students, are a special type of users. They can teach different classes. The Teaches relationship is made with class and not course. This is because it is possible to have many teachers for one course. For instance, there may be one teacher giving a lecture and another teacher in charge of tutorials and labs.

6.3 Courses

Figure 7 focuses on how a course's information is listed in the database.

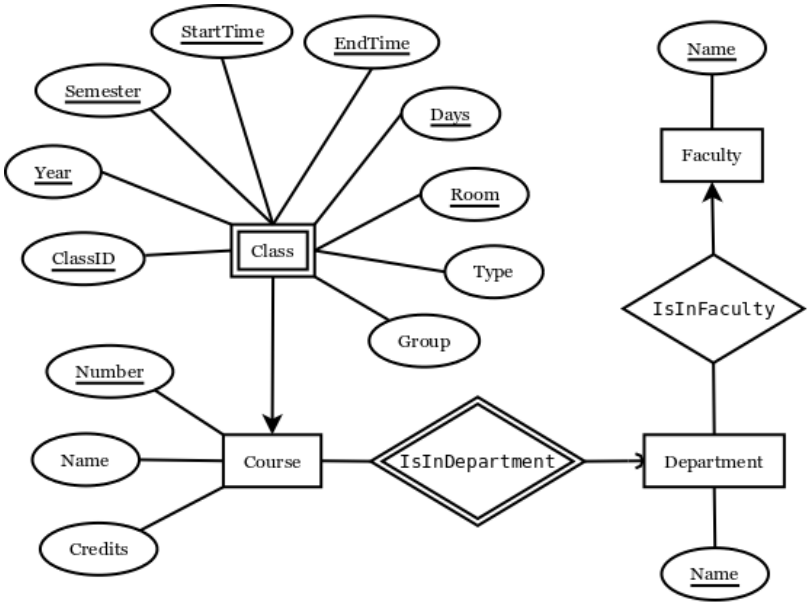

Figure 7 - Courses in the database

6.3.1 Course

A course is uniquely identified by its department and course number. For this reason, each course must be related to Department with the IsInDepartment relationship. The IsInDepartment relationship uses short four letter department names such as COMP or SOEN. The Name attribute of Course is simply the full name describing the course. The number of credits is also stored within the course.

6.3.2 Class

For each course there may be a different number of classes. A class is a weak entity, as it depends on a course to be uniquely distinguished from other classes. This is where you find important information such as semester, year, time, day(s), group and room. The type attribute is used to identify the class as being a lecture, a tutorial or a lab. Because of the large number

of attributes required to uniquely identify a class, an extra attribute has been added: ClassID. The ClassID is simply a convenience in order to relate teachers to classes.

6.3.3 Department

A department is one of the departments by which a course is given. Examples of this are COMP (Computer Science) and SOEN (Software Engineering). The four letter abbreviation is used to relate courses to departments, but the Department entity also stores the full name of the department. As departments are part of a faculty, each department is related to the faculty it belongs to.

6.3.4 Faculty

Each faculty has an abbreviation of a few letters that is used to relate faculties to departments, such as ENCS with COMP. Like departments, the Faculty entity stores the full name of the faculty. For instance, ENCS stands for Engineering and Computer Science.

6.4 Requirements

Figure 8 depicts how the prerequisites for a course are stored in the database.

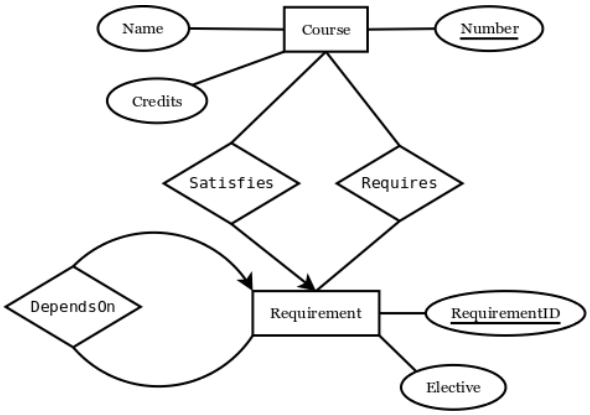

Figure 8 - Representation of requirements.

6.4.1 Simple Requirements

The simplest case of a requirement is a course which is dependent on one or more other courses. For example, let course A be dependent on courses B and C. In the database, course A would be related to a requirement with the Satisfies relationship. That requirement would in turn be related to courses B and C using the Requires relationship.

6.4.2 Elective Requirements

In many programs students have to take at least one elective course. The elective course is usually chosen from a predefined set of possible courses. For example, let course A be dependent on a technical elective. Course A would be related to a Requirement, that would in turn be related to all the possible courses through the Requires relationship. The difference between a requirement where you have to take all of the courses required and one where you only have to pick one of them is made with the Elective attribute. Whenever Elective is not null, it is set as the name of the elective, such as “Technical Elective”.

6.4.3 Multiple Requirements

There are even more complex cases of requirements that require an extra relationship DependsOn that relates a requirement to another requirement. For instance, let course A be dependent on B+C or D+E. This can be represented in the database by an elective requirement depending on two other requirements (B+C and D+E).

6.5 Programs

Figure 9 shows how a program is stored in the database.

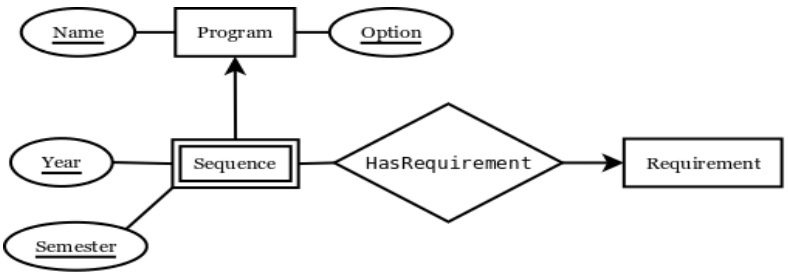

Figure 9 - Representation of a program.

6.5.1 Program Options

A program can have many possible options. However, each option of a program corresponds to a different course sequence. For this reason, a Program is uniquely defined by the program name and its option. An example of a program would be engineering, with an option being software engineering.

6.5.2 Course Sequence

A course sequence is usually presented to the student as a full list of courses to be taken according to each semester of the program. This is why a semester is identified by the Program it is associated with, a year and a semester. The courses that have to be taken to meet the requirements of the sequence are represented using Requirement entities, just like with courses.

7 Process View

The process view of the 4+1 architectural view model examines the process flow of the application. The overall flow is represented by an activity diagram.

7.1 Activity Diagram

Figure 10 shows a UML 2.0 activity diagram representing the process expressed by UC 11 (Generate Schedule Advanced).

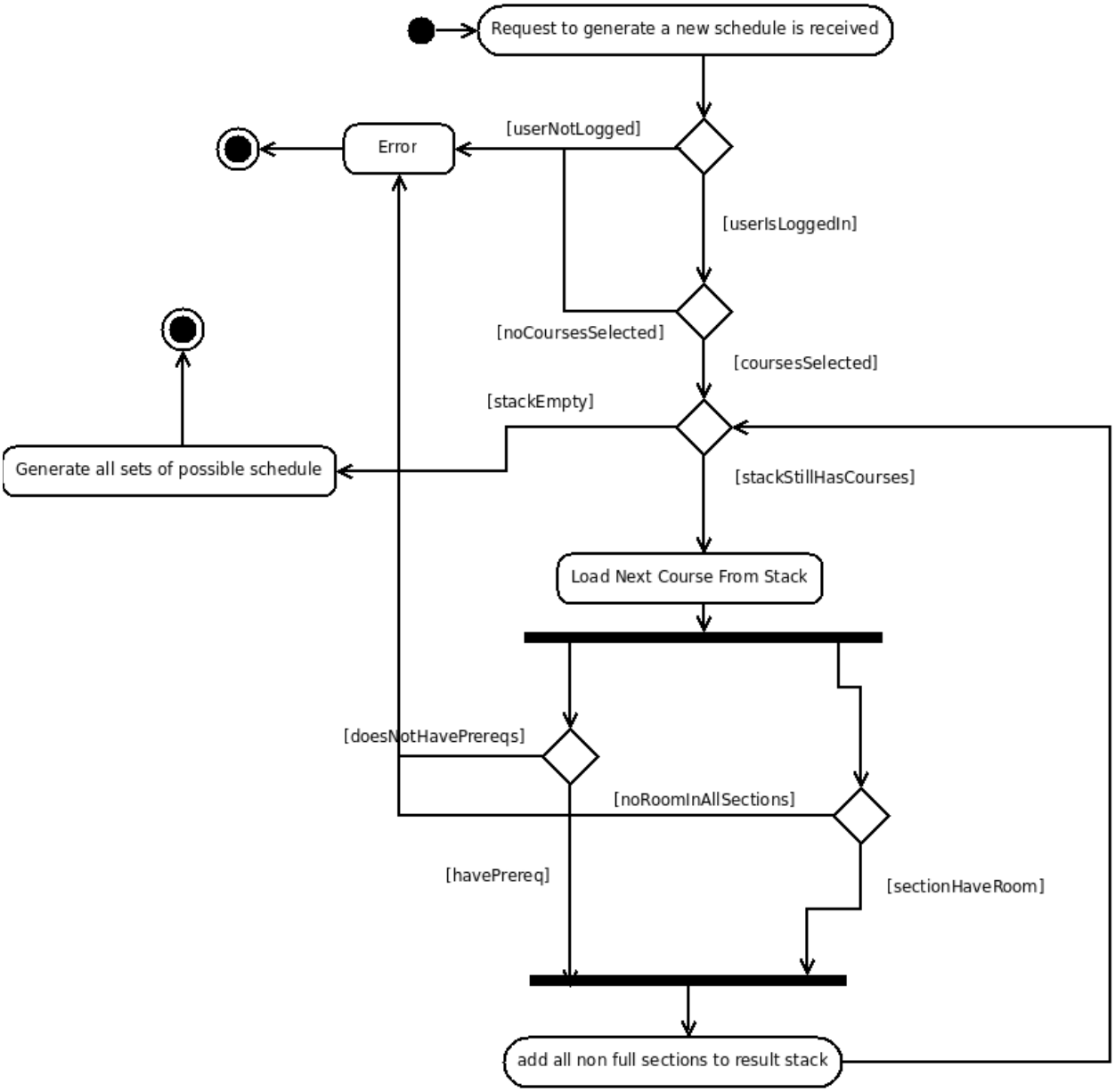

Figure 10 - Activity diagram.

The workflow of UC11 begins when the request to generate a new schedule is received. The login status of the user is verified. If the user is not logged in, an error occurs. This results in an error being given at the output. The front-end is set to deal with it accordingly. This is seen as the end of the workflow for the generate schedule advanced function.

If the user is logged in, the option to select a course becomes available. If no course is selected, an error occurs and the same general process as aforementioned is followed. If a course is selected, it is added to the stack.

If there are courses on the stack, the next course from the stack is loaded. If the student has passed the prerequisites for the given course, and the section is not full, the course is added to the result stack. If the student does not have the necessary prerequisites and/or the section is full, an error occurs and the workflow ends.

Once the stack has been emptied, and no errors have occurred, all of the possible schedules are generated and the workflow ends.

8 Physical View

The physical view of the 4+1 architectural view model examines how each component of the application interacts with the systems it is being run on and by. This is depicted by a deployment diagram.

8.1 Deployment Diagram

Figure 11 shows the deployment diagram for the PlanZilla web application.

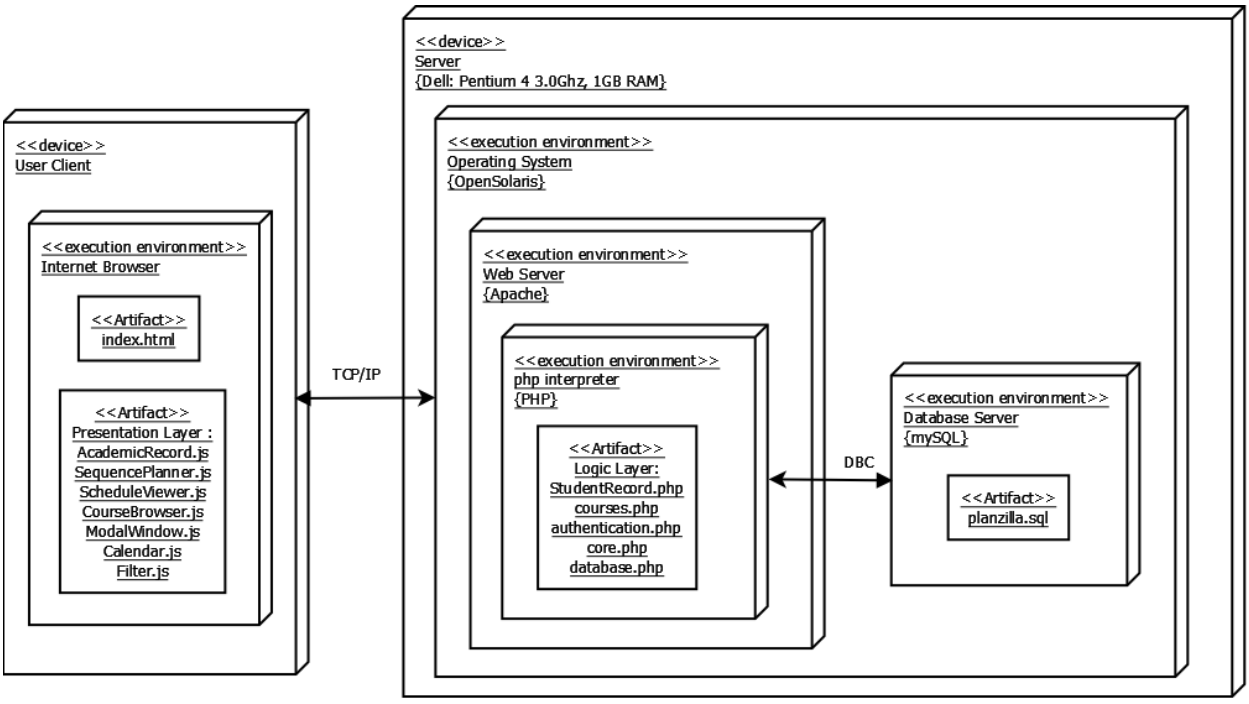

Figure 11 - Deployment diagram.

The deployment of Planzilla's architecture is simple and straightforward. By using this layout, there should be less data being sent between the client and server. This means that the web page will load much faster.

8.1.1 OpenSolaris Server

Two physical device entities are required for this application. One of them is physical server. This server is running on a Dell Pentium 4, which has 1 GB of RAM and a processor speed of 3.0GHz. This computer uses an OpenSolaris platform.

Within the OpenSolarsis Server there are three virtual environments. These environments communicate between each other and execute specific tasks.

8.1.1.1 Apache

The first environment is the web server application, Apache. This layer receives information from the client and processes it. It then selects the PHP script to be executed and sends the command for execution to the PHP interpreter.

8.1.1.2 PHP Interpreter

The PHP interpreter is the second virtual environment. It contains all of the logic tier code written in PHP, which has been separated into classes. This layer communicates with the database as needed.

8.1.1.3 Database Server

The third virtual environment is the database. Its function is to perform database operations such as querying for information or modifying the database. These functions are controlled by the PHP interpreter.

8.1.2 User Client

The other physical entity in the architecture is the actual user client. It contains the Internet browser. In order to optimize the speed and efficiency of the Planzilla application, the presentation layer is processed by the client's browser. Initially, the web server uploads all of the necessary JavaScript files to the client. When the user clicks on a link or inputs data, the only data being retrieved is the data from the database, such as course related data or scheduling information. This is then rendered by the JavaScript and is displayed to the user.

9 Logical View

The logical view of the 4+1 architectural view model shows the breakdown of each component of the application from the

9.1 Class Diagram

Figure 12 shows the class diagram for the PlanZilla application.

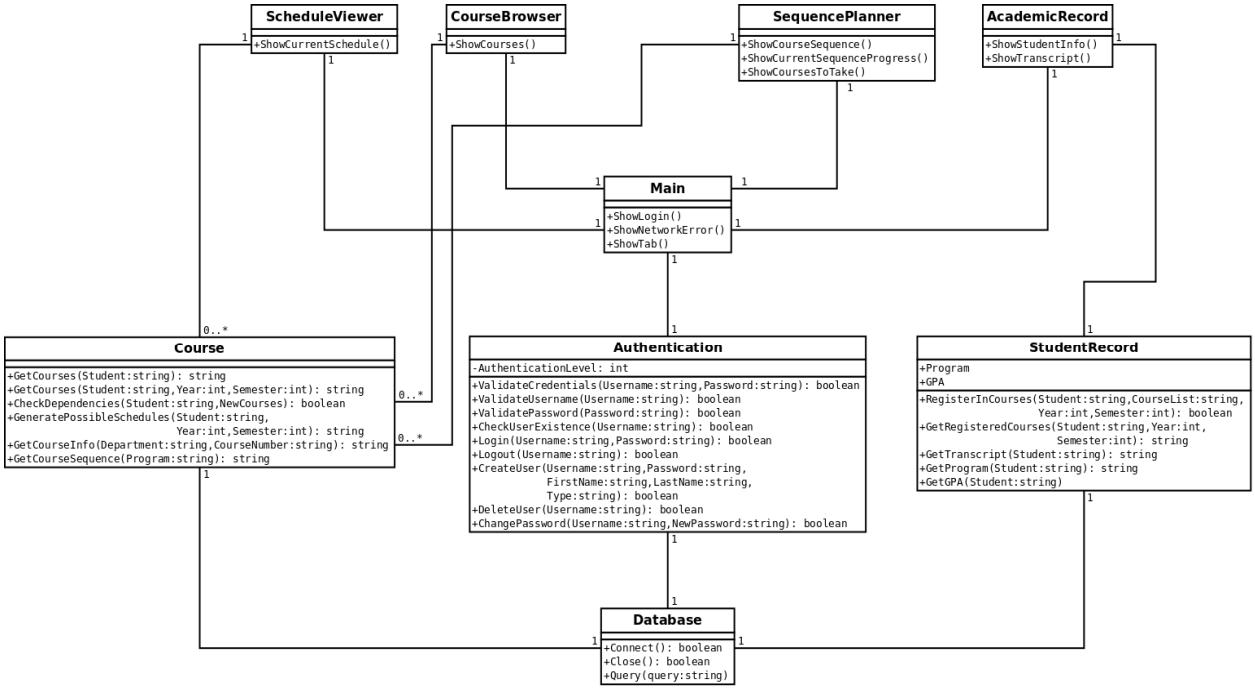

Figure 12 - Class diagram.

9.2 Unit Descriptions

The following section is a detailed description of each class seen in Figure 12. This includes the purpose of the class and its functions and attributes. These classes make up the logic tier.

9.2.1 ScheduleViewer

The SceduleViewer class is to be written in JavaScript as it is part of the presentation tier. As such, its attributes and functions are not listed in this diagram. This class will be used to display a schedule to the user.

9.2.2 CourseBroswer

The CourseBrowser class is to be written in JavaScript as it is part of the presentation tier. As such, its attributes and functions are not listed in this diagram. This class will be used to display a course's information to the user.

9.2.3 SequencePlanner

The SequencePlanner class is to be written in JavaScript as it is part of the presentation tier. As such, its attributes and functions are not listed in this diagram. This class will be used to display the scheduling options to the user.

9.2.4 AcademicRecord

The AcademicRecord class is to be written in JavaScript as it is part of the presentation tier. As such, its attributes and functions are not listed in this diagram. This class will be used to display the student's academic record.

9.2.5 Main

The Main class is the central module of the application. It handles each request from the client and redirects them to the appropriate module, whose task it is to handle the request. This class implements the Core component. It acts as the central node of the program, by loading different tabs presented to the user which will in turn forward requests to the logic tier, all of which passes through the Main module.

9.2.5.1 Reminders

It is important to determine if this design could have any performance issues, such as with the case where many users are connected to a single server. It should also be ensured that Ajax is being used correctly from the client's side, in order to avoid sending unnecessary information to the server.

9.2.6 Course

The Course class is responsible for obtaining course information, verifying any potential course dependencies, and generating possible schedules given any constraints.

9.2.6.1 Important Notes

In order to implement the CheckDependecies() function, an efficient algorithm that will execute the task without wasting too much CPU time on the server must be found.

9.2.7 Authentication

The Authentication class is the module responsible for validating, creating and modifying users. It is an important module, since many of the other classes communicate with this module before proceeding with their function.

9.2.7.1 Reminders

It would be beneficial to create a simple SSL/TLS certificate. This would protect a client's session from hackers attempting to steal their unique ID as a session is opened.

Furthermore, each input from the client needs to be checked for potential security vulnerabilities. For example, SQL injections, which enables a user to potentially manipulate the instructions sent to the database.

9.2.8 StudentRecord

The StudentRecord class handles the tasks of obtaining and modifying information in the student record. This includes retrieving a list of currently registered courses as well as past courses.

9.2.9 Database

The database class abstracts the database and controls access to it. The Query() function receives the SQL query from other classes and it sends it to the MySQL database that's being used for our project. This abstraction was created in order to simplify the maintenance and modification processes.

10 Module Interface Specification

10.1 Authentication Class

10.1.1 Detailed Description

The Authentication class is the module used for validating, creating or modifying users.

10.1.2 Public Member Functions

The following are the public member functions for the Authentication class:

- **ValidateCredentials** ($Username, $Password)
- **ValidateUsername** ($Username)
- **ValidatePassword** ($Password)
- **CheckUserExistence** ($Username)
- **Login** ($Username, $Password)
- **Logout** ($Username)
- **CreateUser** ($Username, $Password, $FirstName, $LastName, $Type)
- **DeleteUser** ($Username)
- **ChangePassword** ($Username, $NewPassword)

10.1.3 Public Attributes

The following is a list of public attributes for the Authentication class:

- **$AuthenticationLevel**
  User authentication level.

10.1.4 Member Function Documentation

Authentication::ChangePassword ($ *Username*, $ *NewPassword*)
Parameters: *string* $Username *string* $NewPassword **Returns:** boolean

Authentication::CheckUserExistence ($ *Username*)

Parameters:

string $Username

Returns:

boolean

Authentication::CreateUser ($ *Username*, $ *Password*, $ *FirstName*, $ *LastName*, $ *Type*)

Parameters:

string $Username
string $NewPassword
string $FirstName
string $LastName
string $Type

Returns:

Boolean

Authentication::DeleteUser ($ *Username*)

Parameters:

string $Username

Returns:

boolean

Authentication::Login ($ *Username*, $ *NewPassword*)

Parameters:

string $Username
string $NewPassword

Returns:

boolean

Authentication::Logout ($ *Username*)

Parameters:

string $Username

Returns:

boolean

Authentication::ValidateCredentials ($ *Username*, $ *NewPassword*)

Parameters:

string $Username
string $NewPassword

Returns:

boolean

Authentication::ValidatePassword ($ *NewPassword*)

Parameters:

string $NewPassword

Returns:

boolean

Authentication::ValidateUsername ($ *Username*)

Parameters:

string $Username

Returns:

boolean

10.1.5 Member Data Documentation

Authentication:: $AuthenticationLevel

User authentication level:

- -1 = Not authenticated.
- 0 = Administrator.
- 1 = Teacher.
- 2 = Student.

10.1.6 Source File

The documentation for this class was generated from the following file:

- www/php/Authentication.php

10.2 Course Class

10.2.1 Detailed Description

The Course class is the module for obtaining course information, verifying dependencies, and generating possible schedules in function of given constraints.

10.2.2 Public Member Functions

The following are the public member functions for the Course class:

- **GetCourses** ($Student)
- **GetCourses** ($Student, $Year, $Semester)
- **CheckDependencies** ($Student, $NewCourses)
- **GeneratePossibleSchedules** ($Student, $Year, $Semester)
- **GetCourseInfo** ($Department, $CourseNumber)
- **GetCourseSequence** ($Program)

10.2.3 Member Function Documentation

Course::CheckDependencies ($ *Student*, $ *NewCourses*)

Parameters:

string $Student
string $NewCourses

Returns:

boolean: True if dependencies are satisfied. False otherwise.

Course::GeneratePossibleSchedules ($ *Student*, $ *Year*, $ *Semester*)

Parameters:

string $Student
int $Year
int $Semester

Returns:

string: An array of possible schedules.

Course::GetCourseInfo ($ *Department*, $ *CourseNumber*)

Parameters:

string $ Department
string $ CourseNumber

Returns:

string: Course information.

Course::GetCourses ($ *Student*, $ *Year*, $ *Semester*)

Parameters:

string $Student
int $Year
int $Semester

Returns:

string: Student's courses for a given semester (may be a past semester).

Course::GetCourses ($ *Student*)

Parameters:

string $Student

Returns:

string: Student's past, present and future courses.

Course::GetCourseSequence ($ *Program*)

Parameters:

string $Program

Returns:

string: Course sequence.

10.2.4 Source File

The documentation for this class was generated from the following file:

- www/php/Course.php

10.3 Database Class

10.3.1 Detailed Description

The Database class is the module for abstracting the database and controlling access to it.

10.3.2 Public Member Functions

The following are the public member functions for the Database class:

- **Connect**()
  Connect to database using credentials from configuration file.
- **Close**()
  Close current connection to the database.
- **Query**()
  Send a query to the database.

10.3.3 Member Function Documentation

Database::Close()

Close current connection to the database.

Returns:

boolean: True if successful. False otherwise.

Database::Connect()
Connect to database using credentials from configuration file. **Returns:** boolean: True if successful. False otherwise.

Database::Query()
Send a query to the database. **Returns:** boolean: True if successful. False otherwise.

10.3.4 Source File

The documentation for this class was generated from the following file:

- www/php/Database.php

10.4 StudentRecord Class

10.4.1 Detailed Description

The StudentRecord class is the module for obtaining or modifying the student record information.

10.4.2 Public Member Functions

The following are the public member functions for the StudentRecord class:

- **RegisterInCourses** ($Student, $CourseList, $Year, $Semester)
- **GetRegisteredCourses** ($Student, $Year, $Semester)
- **GetTranscript** ($Student)
- **GetProgram** ($Student)
- **GetGPA** ($Student)

10.4.3 Private Attributes

The following are the private attributes for the StudentRecord class:

- **$Program**
- **$GPA**

10.4.4 Member Function Documentation

StudentRecord::GetGPA ($ *Student*)

Parameters:

string $Student

Returns:

int: Student's GPA.

StudentRecord::GetProgram ($ *Student*)

Parameters:

string $Student

Returns:

string: Student's program.

StudentRecord::GetRegisteredCourses ($ *Student*, $ *Year*, $ *Semester*)

Parameters:

string $Student

int $Year

int $Semester

Returns:

string: Courses in which the given student is registered for a particular semester.

StudentRecord::GetTranscript ($ *Student*)

Parameters:

string $Student

Returns:

string: Student's transcript.

StudentRecord::RegisterInCourses ($ *Student*, $ *CourseList*, $ *Year*, $ *Semester*)

Parameters:

string $Student

string $CourseList

int $Year

int $Semester

Returns:

boolean: True if registration is successful. False otherwise.

10.4.5 Member Data Documentation

StudentRecord::$GPA [private]

Student's GPA

StudentRecord::$Program [private]

Student's Program

10.4.6 Source File

The documentation for this class was generated from the following file:

- www/php/StudentRecord.php

11 Scenario View

11.1 Use Case Diagram

Figure 2 shows the use case model for this application. As previously mentioned, there are two actors: a registered user and an unregistered user. The registered user has access to all of the use cases. The unregistered user is limited to the public use cases. More specifically these are opening the application (UC1), logging in (UC2), viewing the schedule of a given room or teacher (UC5) and browsing the course list (UC6). It is important to note that should an unregistered user attempt to login, this should result in a failure end condition since an unregistered user does not have an account.

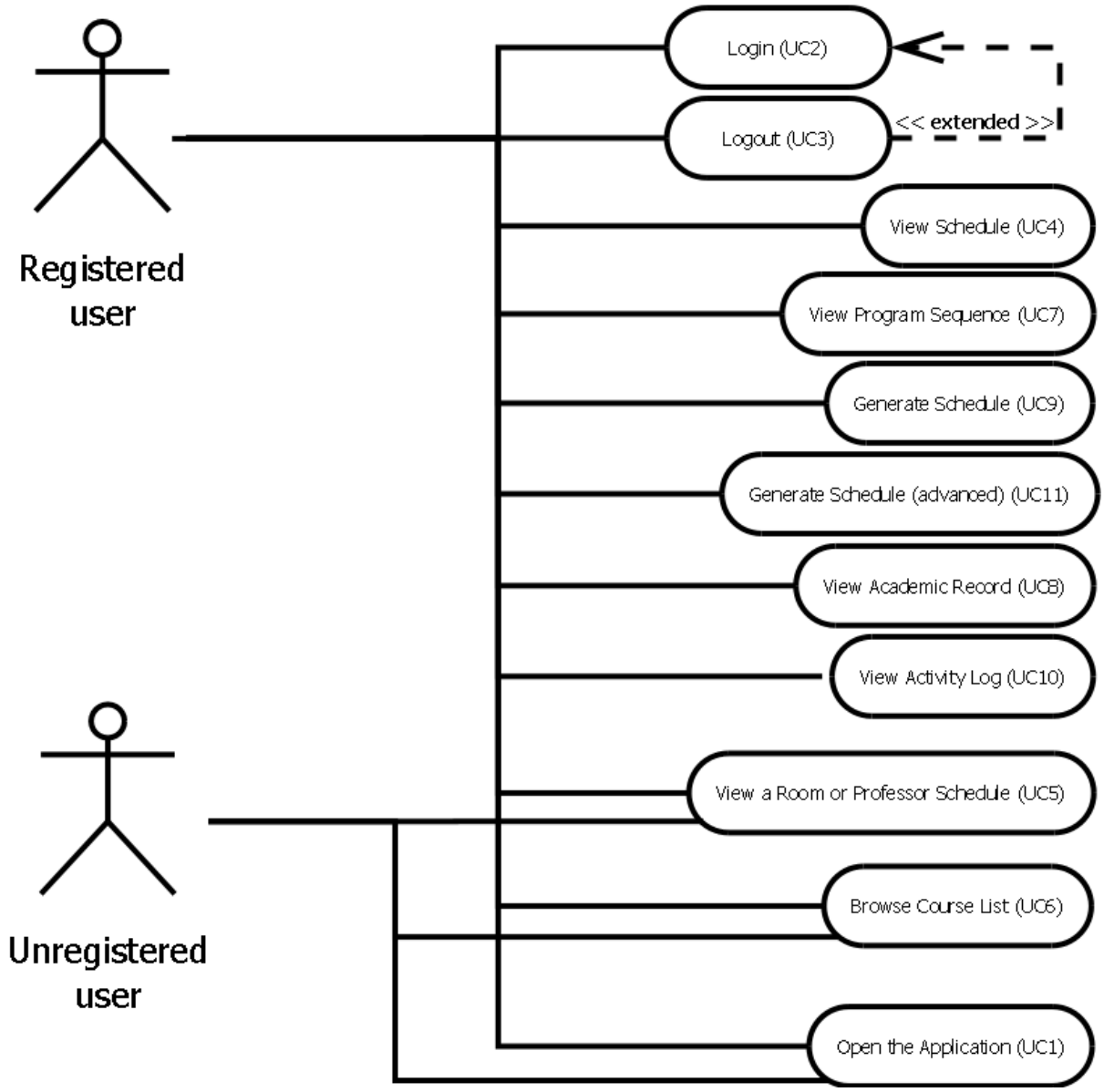

Figure 13 - Use case diagram

12 Dynamic Design Scenarios

12.1 Browse Course List

12.1.1 System Sequence Diagram 1 (SSD1)

Figure 13 shows the system sequence diagram for the Browse Course List use case (UC6).

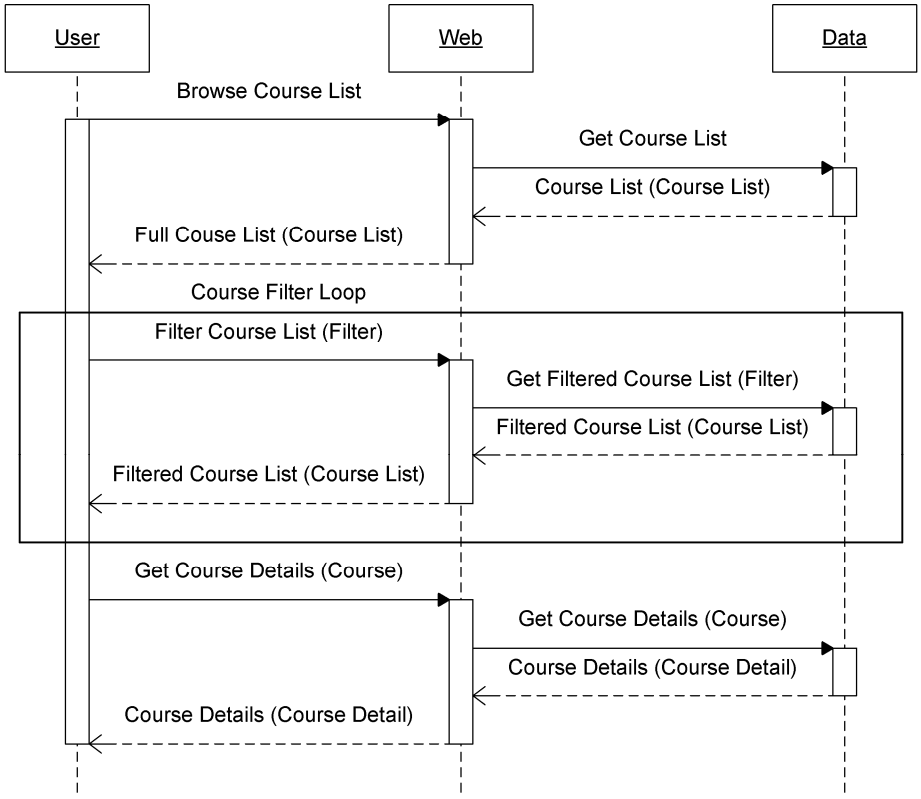

Figure 14 - Browse Course List system sequence diagram (SSD1).

The user activates the "Browse Course List" feature causing the application to query the database for all the courses. This list in then returned to the user. The user may filter the list as often as desired. In doing so, the database is queried for a filtered course list which again is returned to the user. Finally, the user selects a course, which causes the system to query the database for the details on the course and returns the result to the user.

12.1.2 Operational Contract 1.1 (CO1.1)

It is important to note that for SSD1, there is only one operational contract – i.e. DBgetCourses. This function is not limited by a number of courses and can therefore be used to get all courses, get a select few courses or simply get one course.

Name:	DBgetCourses()
Related Use Case:	View Courses, View Sequence, View Schedule, Generate Schedule
Preconditions:	• A table of courses exists in the database
Post Conditions:	♦ List of courses is retrieved from database

12.2 Generate Schedule (Advanced)

12.2.1 System Sequence Diagram 2 (SSD2)

Figure 14 shows the system sequence diagram for the Generate Schedule (Advanced) use case (UC11).

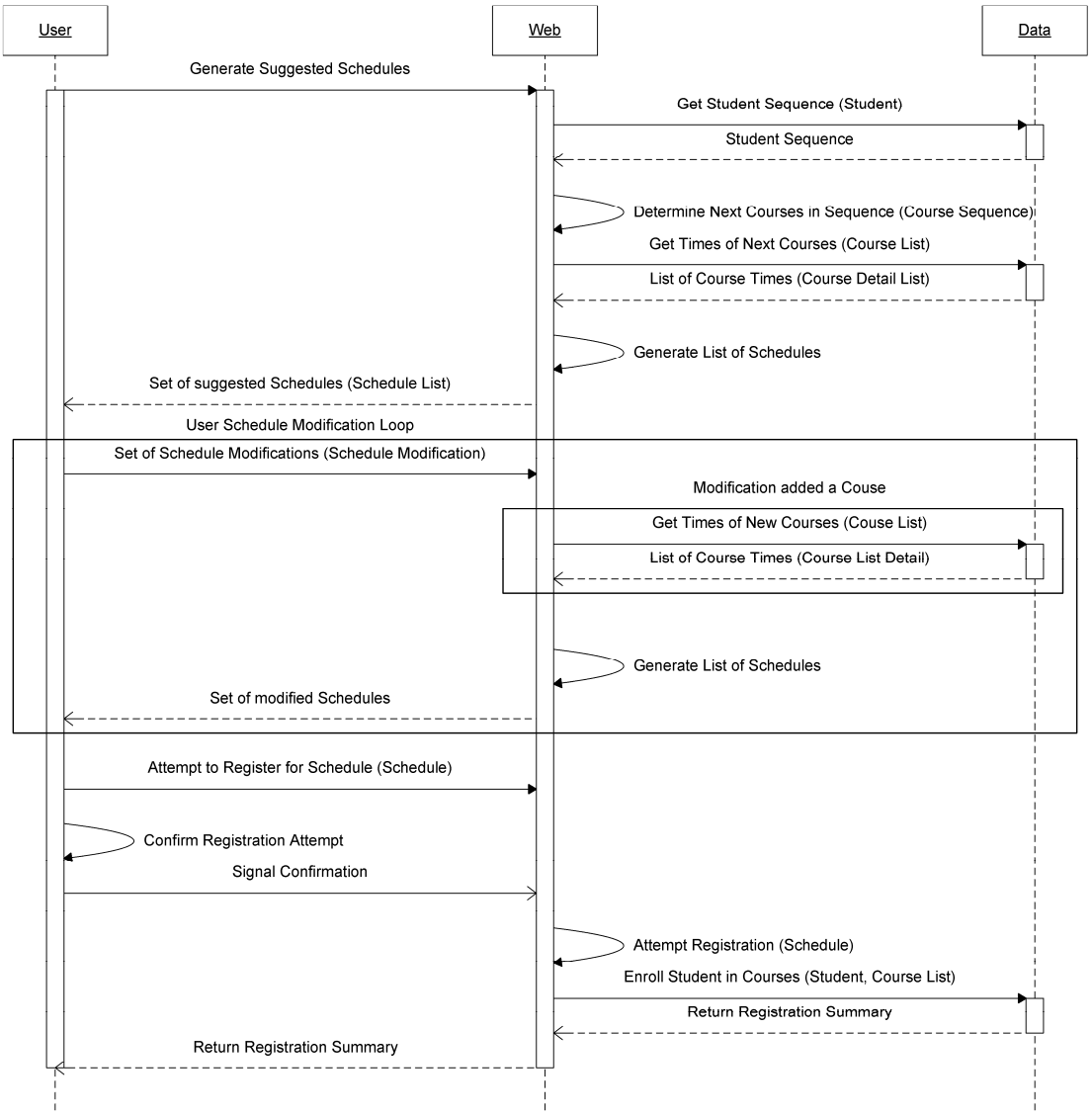

Figure 15 - Generate Schedule (Advanced) system sequence diagram (SSD2).

The user activates the "Generate Schedule" feature causing the application to query the database for the student's sequence. The sequence is then used to determine the courses the student must take next. The database is queried for the timeslots of these courses. All of the possible schedules generated using these courses are then returned to the user.

The user may make modifications to the suggested list of courses as often as desired. Upon doing so, the database is queried for the timeslots of any new courses selected. A new set of generated schedules in returned to the user. The user then selects one and confirms it.

The confirmation is sent to the system, which then attempts the registration and records it in the database. The database returns the summary of all registrations and failures - if any – to the application and the application returns this summary to the user.

12.2.2 Operational Contract 2.1 (CO2.1)

Name:	**DBgetStudentRecord()**
Related Use Case:	View academic record
Preconditions:	• User is a valid student logged in to the system
Post Conditions:	♦ Student record of user is retrieved from the database

12.2.3 Operational Contract 2.2 (CO2.2)

Name:	**DBgetGPA()**
Related Use Case:	View academic record
Preconditions:	• User is a valid student logged in to the system
Post Conditions:	♦ GPA of the user is retrieved from the database

12.2.4 Operational Contract 2.3 (CO2.3)

Name:	**DBgetSequence()**
Related Use Case:	View Sequence, Generate schedule
Preconditions:	• A table of course sequences is in the database
Post Conditions:	♦ Course sequence data is retrieved from database

12.2.5 Operational Contract 2.4 (CO2.4)

Name:	**genJSONSchedList()**
Related Use Case:	View Courses, View Sequence, View Schedule, Generate Schedule
Preconditions:	• User is logged in to the system • Student Record is available for the user • User has requested a schedule to be generated • Course sequence data is available

	• Course data is available
Post Conditions:	♦ List of schedules and courses is generated in JavaScript Object Notation (JSON)

12.2.6 Operational Contract 2.5 (CO2.5)

Name:	**renderSchedule()**
Related Use Case:	View Courses, View Sequence, View Schedule, Generate Schedule
Preconditions:	• A list of schedules was generated was provided
Post Conditions:	♦ Markup is presented to the user displaying possible schedules

12.2.7 Operational Contract 2.6 (CO2.6)

Name:	**applyFilter()**
Related Use Case:	Generate Schedule
Preconditions:	• User is logged in to the system • User has selected some conditions to filter possible schedules
Post Conditions:	♦ The number of possible schedules with applied constraints is returned

12.2.8 Operational Contract 2.7 (CO2.7)

Name:	**DBcheckSpots()**
Related Use Case:	Generate Schedule
Preconditions:	• User is logged in to the system • User is a valid student • User selects a valid course to check free spots • There exists a table in the database with the number of seats in the section
Post Conditions:	♦ The number of free seats for the selected section is returned

12.2.9 Operational Contract 2.8 (CO2.8)

Name:	**ValidateRegRequest()**
Related Use Case:	Generate Schedule
Preconditions:	• User is logged in to the system • User is a valid student • User attempts to register for a course
Post Conditions:	♦ Allows student to register for course if validated ♦ Throws an error if student cannot register due to restrictions

12.2.10 Operational Contract 2.9 (CO2.9)

Name:	DBregisterStudent()
Related Use Case:	Generate Schedule
Preconditions:	• User is logged in to the system • User is a valid student • User attempts to register for a course • Database table exists for students registered in courses • Registration request was valid
Post Conditions:	♦ Student is registered in selected course

12.3 Miscellaneous

This section covers the operational contracts that are relevant to both system sequence diagrams.

12.3.1 Operational Contract 3.1 (CO3.1)

Name:	validateUsername()
Related Use Case:	Login
Preconditions:	• User attempts to log on to the system
Post Conditions:	♦ A valid user name was provided by the user ♦ An invalid name is provided

12.3.2 Operational Contract 3.2 (CO3.2)

Name:	validatePwd ()
Related Use Case:	Login
Preconditions:	• User attempts to log on to the system
Post Conditions:	♦ A valid password is provided ♦ An invalid password is provided

12.3.3 Operational Contract 3.3 (CO3.3)

Name:	Auth()
Related Use Case:	Login
Preconditions:	• Valid user name and passwords were provided by the user
Post Conditions:	♦ User is authenticated to the system ♦ User has entered incorrect logon information and is not authenticated

12.3.4 Operational Contract 3.4 (CO3.4)

Name:	DBgetUsers()
Related Use Case:	Login
Preconditions:	• Database table of users account data exists
Post Conditions:	♦ User account data is read from the database

12.3.5 Operational Contract 3.5 (CO3.5)

Name:	generateToken()
Related Use Case:	Login
Preconditions:	• User has logged on to the system and has been authenticated
Post Conditions:	♦ A token is generated for the unique identification of a user

12.3.6 Operational Contract 3.6 (CO3.6)

Name:	giveCookie()
Related Use Case:	Login
Preconditions:	• Authenticated user is logged on to the system • User's browser supports cookies • A token has been generated for the user
Post Conditions:	♦ The user's browser is given the cookie to identify them ♦ No cookie is given

12.3.7 Operational Contract 3.7 (CO3.7)

Name:	validateCookie()
Related Use Case:	Login
Preconditions:	• User logs on to system • User has a token from a past log on • User's browser provides cookie info to the system
Post Conditions:	♦ User with a valid token is authenticated ♦ User with an invalid token is still required to authenticate

12.3.8 Operational Contract 3.8 (CO3.8)

Name:	displayError()
Related Use Case:	ALL
Preconditions:	• An error is thrown
Post Conditions:	♦ Error is displayed to the user

12.3.9 Operational Contract 3.9 (CO3.9)

Name:	**logEvent()**
Related Use Case:	ALL
Preconditions:	• A meaningful state change has occurred in the system
Post Conditions:	♦ The state change is logged and stored in the database