6 SERVER SETUP

6.1 Assembla.com Team Collaboration Space

Assembla.com is a website that offers an integrated solution for online team collaboration. While various paid plans are available, a free public plan exists that provides enough resources for the scope of this project.

6.1.1 Team Collaboration

In order to manage a project, the team leader can create a new team collaboration space and invite each member to join it. This requires that each member of the team create an account on assembla.com. To simplify things, we have used our Concordia netnames as our account names. Through team collaboration space, members can easily reach the rest of the team through a messaging system and a chat function. There is also a place where one can view a history of the latest events.

6.1.2 Git Distributed Version Control System

One of the main reasons for using assembla.com's service is because they provide free version control system hosting. Version control can easily be added to the team collaboration space by the administrator with a one-click installation. Assembla.com provides multiple version control systems such as subversion, mercurial and git. While subversion is a popular version control system, the new trend is to move to git. Git is a more recent and efficient system originally created by Linus Torvalds. When compared to subversion, git is a faster and more efficient system, especially in regards to branching.

6.1.3 Webhook

Another nice feature that made assembla.com a good choice for hosting our git repository is webhooks. The webhook is a simple HTTP POST that is sent automatically to notify of an event. As we wanted to host our web application on a separate server, we are using it to trigger an update of the application on the remote server whenever someone commits changes on the git repository. Without webhooks, this would have to be done by constantly polling the git server for updates, which is a quite inefficient way of doing things.

6.2 OpenSolaris Server

OpenSolaris is a free open source operating system based on Sun's Solaris. It is one of the variants of UNIX based on SVR4. As Sun mostly ships Solaris with their selection of high performance servers of all types, OpenSolaris is a good choice for making our web server. For this project, a dedicated server with a P4 3.0GHz and 1 GB of RAM is used to host our web application in development.

6.2.1 Web Stack (AMP)

OpenSolaris' web stack, called the "AMP" stack, which stands for Apache, MySQL and PHP, provides an easy way to set up the various programs needed to host our web application. This is perfect for us since this is exactly what we need.

6.2.2 Development Web Stack (AMP-dev)

A web development stack is also available, providing a full set of tools to develop web applications. While it is not really an option for team members to use, since nobody uses OpenSolaris as their desktop OS, it comes with documentation and instructions on how to use the tools it installs. This is great as it provided a lot of ideas on what alternatives to use, such as the NetBeans IDE.

6.2.3 Hostname

The server is connected to the internet with a residential Videotron connection. Since Videotron uses the usual HTTP port 80, we are using the alternative port 8080. Also, since Videotron does not provide a static IP, we registered a free hostname from dyndns.org. It can be automatically updated to point to the current IP address of the server using a client program. Our website can be accessed at http://team7.ath.cx:8080/

6.2.4 Individual Test Space

Since we are eight people working on the same website, a separate space is provided to each member so they can do their own testing without breaking the work of other members. Each member has an account on the server with a username corresponding to their Concordia netname. Apache has been configured to enable the user directory features, which gives each member a web folder accessible at http://team7.ath.cx:8080/~username/. Each user's home folder contains a public_html folder where they can put their files for their personal web space.

7 DEVELOPER CLIENT SIDE SETUP

7.1 Git

7.1.1 Installation (Windows)

A windows port of git is available as the msysgit project, available at http://code.google.com/p/msysgit/. The installation is straightforward – just follow the installation wizard and a shortcut should be added on the desktop. The program gives a full bash shell with git installed.

7.1.2 Installation (Linux / Mac OS X)

On Linux, the distribution's package manager is able to install git. On Mac OS X, an installer is available at http://code.google.com/p/git-osx-installer/.

7.1.3 Initial Configuration

Log in to your assembla.com account, click on the Source/Git tab. Click on "More Instructions" under "clone/push URL". There are two URLs that can be used for git, one is for public read-only access, and the second one is for read/write access. As we want read/write access, we need to use git@git.assembla.com:team7.git. The first thing that you have to do is to set up ssh with a public key that you need to give to assembla.com so that they can identify you. If you already have a key, it should normally be placed in ~/.ssh/id_rsa.pub. If you do not have a key, you can generate a new one with "ssh-keygen –t rsa". The ida_rsa.pub file is just a text file containing your public key. Either output it to your terminal to copy it with "cat ~/.ssh/id_rsa.pub" or open it with a text editor. In assembla, click on "My Start Page" on top right hand corner and then click the "Profile" page. Scroll down the profile page, and there should be a space to paste your ssh public key. Paste it there and save your changes. It is important that this step is correctly done otherwise you will not be able to modify anything on the git repository.

Once the ssh keys are correctly set up, open a terminal (or in Windows, start the git bash shell) and then enter the command "git clone git@git.assembla.com:team7.git". The result should resemble Figure 7.1.3-1.

```
marcan_m@linux:~/$ git clone git@git.assembla.com:team7.git
Initialized empty Git repository in /home/marcan_m/team7/.git/
remote: Counting objects: 49, done.
remote: Compressing objects: 100% (44/44), done.
Receiving objects: 100% (49/49), 78.10 KiB | 10 KiB/s, done.
remote: Total 49 (delta 5), reused 0 (delta 0)
Resolving deltas: 100% (5/5), done.
```

Figure 7.1 - Example of proper git clone

If you see something like this, you have successfully fetched a copy of the git repository with full history of the source code. Now switch to the newly created directory using “cd team7”. Once you are inside the directory, you can start configuring settings specific to your own copy of the git repository. “git config -l” lists the current configuration. Figure 7.1.3-2 shows an example of this.

```
marcan_m@linux:~/team7$ git config -l
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
remote.origin.url=git@git.assembla.com:team7.git
branch.master.remote=origin
branch.master.merge=refs/heads/master
```

Figure 7.2 - Example of a list of git configurations

We need to configure two settings that do not appear in the current list of settings: “user.name” and “user.email”. The instructions page on assembla.com should give instructions similar to this. Figure 7.1.3-3 shows how this is done.

```
git config user.name “marcan_m”
git config user.email “marcandre.moreau@gmail.com”
```

Figure 7.3 - Instructions for configuring the user.name and the user.email settings

It is important that these two settings match the username and email address listed in your assembla.com account, otherwise you will fail to submit any changes to the remote git repository. It is not necessary to use the –global flag, since we only need these settings to be set for this git repository. Next, verify that the new settings have been accepted. See Figure 7.1.3-4 for an example.

```
marcan_m@linux:~/team7$ git config -l
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
remote.origin.url=git@git.assembla.com:team7.git
branch.master.remote=origin
branch.master.merge=refs/heads/master
user.name=marcan_m
user.email=marcandre.moreau@gmail.com
```

Figure 7.4 - Instruction for calling the configured settings and a list of the current ones

You should now be ready to start making changes.

7.1.4 Contributing Changes

So you have worked on the project and you are ready to contribute your changes back to the git repository. The first thing you must do is add the files you have modified to the set of modifications you want to submit. Adding a file is as simple as typing “git add <filename>”. Removing a file is also simple, with “git rm <filename>”. If you want to move a file without losing its history in the repository, you can use “git mv <source> <destination>”. These are the most basic commands you need to know. Once you’ve added all of the items you wanted to submit, you can use “git status” to get a summary of what is going to be added and what is going to be ignored. Figure 7.1.4-1 shows an example where two new files have been created since the last version, but only one of the will be submitted:

```
marcan_m@linux:~/team7$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#        new file:   a.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#        b.txt
```

Figure 7.5 - Example of a git status summary

Once you are ready, you can use commit these changes. Figure 7.1.4-2 shows an example of the instruction for this.

```
Marcan_m@linux:~/team7$ git commit –a –m “comment describing the changes”
```

Figure 7.6 - Instruction for a git commit including a comment

A git commit creates a new version and commits the changes into that new version. The “-a” commits all of the added changes, and the “–m” is used to include a comment describing what those changes are. You can use git commit as many times as you want without affecting the work of other people, since the new version is only local to your machine at this stage.

After using git commit, you need to “push” your changes to the remote git repository so that the other developers can get it. Figure 7.1.4-3 shows the command for this.

```
marcan_m@linux:~/team7$ git push origin master
```

Figure 7.7 - Instruction for pushing changes into the git repository

The “origin” command is the root of the remote git repository, and the “master” command links to the master branch. Because this is a small project, we will not be using the git superb branching feature. W e will only use a single branch, a.k.a. the master branch.

7.1.5 Synchronizing Work

If you attempt a “git push” and it fails, chances are it is because someone else has pushed a new version that is more recent than the one your work was based on. In this case, use “git pull” to fetch the latest version from the git repository. In the case that you had made changes that have not yet been contributed, git will automatically merge your changes with the latest version. After a successful merge, you should be ready to attempt to push your changes again. To see what the changes were, you can use “git log” or use a graphical tool like gitk. The log of the latest changes can also be browsed from assembla.com.

7.2 NetBeans IDE

7.2.1 Installation

On Linux, NetBeans can be installed by the distribution’s package manager. On Mac OS X and Windows, installers are available from netbeans.org. You need to install the PHP NetBeans bundle. If you choose to install the full NetBeans bundle, you will also need to install the JDK (Java Development Kit).

7.2.2 Plugins

In NetBeans, click on “tools->plugins” to get a list of available plugins, or to get updates. If you installed NetBeans without PHP, you can install the PHP plugin from there. A Javascript debugger plugin is available, which may be helpful.

7.2.3 Initial Configuration

Create a new project by going to “File->New Project” and selecting “PHP/PHP Application with Existing Sources”. Click “next”, which brings you to a dialog asking you for the source files location. Browse for the “www” folder of your local git repository. Call the project “Team7”, and then click “next”. For “Run As”, select “Remote Web Site (FTP, SFTP)”. Change the project url to http://team7.ath.cx:8080/~username/, where you replace “username” with your own, and a trailing slash at the end. For “remote connection”, click “manage”. Create a new connection of type SFTP. The hostname is team7.ath.cx, with your own username and password for ssh. Set the initial directory to /export/home/username/public_html where “username” is again your own username. Save the new connection by clicking “OK” and then select it. Empty the “Upload Directory” field and set “Upload Files” to “On Run”. Click “Finish” and you are done for that part. You can now code PHP in NetBeans. Clicking the green play button will automatically deploy your PHP code in your personal web folder and open a browser for you.

7.2.4 Remote PHP Debugger

In order to be able to use the remote PHP debugger from NetBeans, you need to open an ssh tunnel for port 9000 from the server to your own machine. This has to be done because the debugger attempts to connect to the debugger client locally on port 9000. By redirecting port 9000 to your own machine instead, the debugger will be able to connect to the PHP debugger client embedded into NetBeans. When you wish to use that feature, open a terminal and open an ssh tunnel as shown in Figure 7.2.4-1.

```
ssh -R 9000:localhost:9000 username@team7.ath.cx
```

Figure 7.8 - Command for opening an ssh tunnel in a linux terminal.

Leave the terminal window open for the entire time you need to use the remote PHP debugger.

8 PROGRAMMING LANGUAGES AND LIBRARIES

8.1 Languages

8.1.1 PHP

PHP is a server-side scripting language, which will be used to interface the database with the client-side application as well as process client-side requests with database information. We choose PHP to code the logic tier of the application because of several factors. It is the language that the largest number of people in our team felt comfortable coding with. It is well documented, largely used in the wild and easily deployable on any Apache instance. The code written using PHP is generally platform and environmentally neutral so it can be ported to any other environment quite easily without costly compilation.

8.1.2 HTML

HTML, which is short for Hypertext Markup Language, is a logical representation of a web interface and will used to organize the client-side aspect of the software. HTML was adopted for the front-end system, in order to comply with some of the platform constraints we met with in the problem statement defining our project. While many different technologies could display information in a web browser, only HTML provide compatibility across the largest amount of platforms and vendors. It is a well-defined, standardized markup language that will make the deployment of our application easier on any HTML 4.01 compliant browser, such as ~~Internet Explorer 6.0+, Safari 2.0+, Chrome 2.0+~~ or Firefox 3.5+.

8.1.3 CSS

CSS, which is short for Cascading Style Sheets, is a system which maps visual styles and themes onto the web interface. Using CSS to style the page, an HTML document need not contain the visual aspect of the content, leaving it only the content and the structure in the HTML document. CSS can be disabled to permit a visually impaired person to access the page with little outside assistance.

8.1.4 Javascript (a dialect of ECMAScript)

Javascript is a client-side scripting language, which will be used to interact with the user on the web interface and send requests to the server and present responses from the server. In order to ease development by having a front-end with a constant and easily maintainable state, we choose JavaScript. Manipulating the object displayed in the browser's window will enable us to minimize the load placed on our logic tier thus complying with requirements of speed and load we have to deal with. Javascript is the standard for interacting with the DOM (domain object model) that the browser exposes, in order to enable a programmer to manipulate objects parsed in the HTML code in various ways. This feature is standard across all major browsers aforementioned and is enabled on more than 95% of the web user's [1].

8.2 Libraries

8.2.1 jQuery

jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript [2]. It is lightweight and offers a developer a lot of options that will help implement a complex UI project like ours. This open source project is maintained by the jQuery foundation, which is an NPO professionally run by a group of software engineers. The most prominent figure is John Resig, founder of jQuery, currently employed at Mozilla Corporation. Companies like Microsoft, Nokia, Google, Dell, Wordpress actively support jQuery. At the time of writing, 26.95% of all web sites used jQuery. Of all the sites that used Javascript as a web technology, more than 37% used jQuery which makes it the most popular technology used on the web today [3]. It is even more popular than Adobe Flash [4]. Of course popularity is of no interest without a community of devoted developers that provide bug fixes and plug-ins written for jQuery. jQuery is being served by google CDN (content delivery network) which ensure an up-to-date version is always running in our production environment [5].

8.2.2 jQueryUI

While jQuery provides an interface between the browser DOM and Javascript, jQueryUI provides a set of rich graphical user interface element that is highly reusable and customizable. We will particularly be interested in jQueryUI's ability to manage color themes, tabs, modal dialog, icons and buttons in a visually consistent manner without any added code in our project. jQueryUI is being served by google CDN (content delivery network), which ensure an up-tp-date version is always running in our production environment.

8.3 Plugins

Plugins are pieces of code maintained by a third-party. We use plugins to alleviate development while help the development process.

8.3.1 jQuery-Week-Calendar

The calendar view is our standard calendar control for displaying to the screen the schedule objects. Our goal is to provide a rich user experience, so an outlook-like view is preferable. This is exactly what the jQuery-Week-Calendar plugin by Rob Monie provides us with [6].

9 OTHER SOFTWARE

The following section covers the software used by the development team in order to plan and produce the application.

9.1 Operating Systems

9.1.1 Microsoft Windows

Microsoft Windows is the most widely used desktop operating system in the world and is one of the target platforms for the client-side aspect of the software being produced. It will be used by the team members mostly for its Office software.

9.1.2 Apple Mac OS X

Apple Mac OS X is a UNIX compliant operating system, which will be used by certain members of the development team to produce parts of the application. It will also be used to test the client-side aspect of the application for compatibility. Its use in the project is strictly a matter of personal preference of some members of the development team.

9.1.3 GNU/Linux

GNU/Linux is a UNIX compliant operating system, which will serve as the back-end of our application. It has been chosen for having a great architecture and a solid reputation as an operating system in the industry. It will be used in conjunction with the Apache server, the mySQL database management system and PHP.

9.2 Office Software

9.2.1 Microsoft Office

Microsoft Office is a set of software applications, which will be used in the planning and the documenting of the application during development. More specifically, the Microsoft Office Word, Microsoft Office Excel and Microsoft Office Visio applications will be used.

9.2.2 Dia

Dia is a software application, which will be used to construct the different diagrams required in documenting key aspects of the application.

9.2.3 MySQL

MySQL is a database management system, which will be used to store all of the persistent information used by the application. It one of the most used database management systems used in web applications and offers excellent performance with nominal overhead.

10 HUMAN RESOURCES

This section details the members of the development team, their roles, their technical experience and their average availability per week.

10.1 Eric Chan

Role:	Team Leader, Programmer
Knowledge:	- Web Development
	- Database management
	- Assembler, C, C++, Perl, Python, HTML/CSS/JavaScript
Experience:	- Team projects
	- Internships
Availability:	4 hours weekly

10.2 Sébastien Parent-Charrette

Role:	Programmer
Skill:	- Database development
	- Low level system programming
	- Assembler, C, C++, Java, Visual Basic, Perl, PHP
Experience:	- Contract based software development
	- Team projects
Availability:	4 hours weekly

10.3 Corey Clayton

Role:	Programmer
Skill:	- Low level system programming
	- Mobile hardware system development
	- Assembler, C, C++, Perl, Python
Experience:	Team projects
Availability:	4 hours weekly

10.4 Mathieu Dumais-Savard

Role:	Programmer
Skill:	- Web Development
	- Database Management
	- HTML/CSS/JavaScript, Perl, PHP, Python
Experience:	- Extensive number of web related personal projects.
	- Business Intelligence
	- Team projects
Availability:	4 hours weekly

10.5 Andreas Eminidis

Role:	Programmer
Skill:	- Database Development
	- Detailed knowledge of security practices
	- Assembler, C/C++, C#, Java, Perl, PHP, Python
Experience:	- Work Experience
	- Team projects
Availability:	4 hours weekly

10.6 Phuong-Anh-Vu Lai

Role:	Programmer
Skill:	- Database Programming
	- 3D graphics programming
	- C/C++, Java, Visual Basic, Perl, PHP, MATLAB
Experience:	- Team projects
	- Quality Assurance Testing for mobile device applications
Availability:	4 hours weekly

10.7 Julia Lemire

Role:	Documentation Coordinator, Programmer
Skill:	- Group management skills
	- C++, C#, HTML/CSS/JavaScript (jQuery, Ext JS)
Experience:	- Work Experience in web application maintenance
	- Team projects
Availability:	4 hours weekly

10.8 Marc-André Moreau

Role:	Systems Administrator, Programmer
Skill:	- Detailed knowledge of networking protocols
	- Detailed knowledge of security practices
	- Knowledge of documentation process
	- Assembler, C/C++, Java, Perl, PHP, Python
Experience:	- Extensive number of personal projects
	- Work Experience (incl. Software development process)
	- Team projects
Availability:	4 hours weekly

11 3-TIER ARCHITECTURE

11.1 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 11.1-1 shows the different components of a 3-tier architecture:

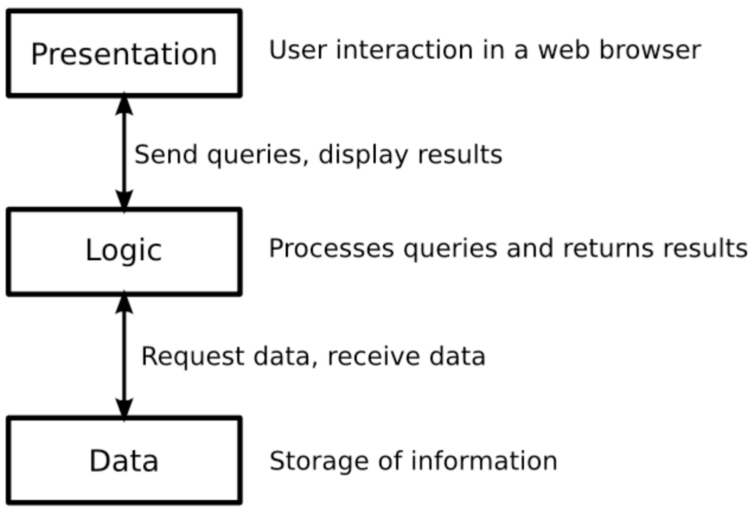

Presentation
User interaction in a web browser
Send queries, display results
Logic
Processes queries and returns results
Request data, receive data
Data
Storage of information

Figure 11.1 - 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.

12 DEVELOPMENT VIEW

12.1 Component Diagram

Figure 12.1 shows a component diagram depicting the 3-tier architecture used for PlanZilla.

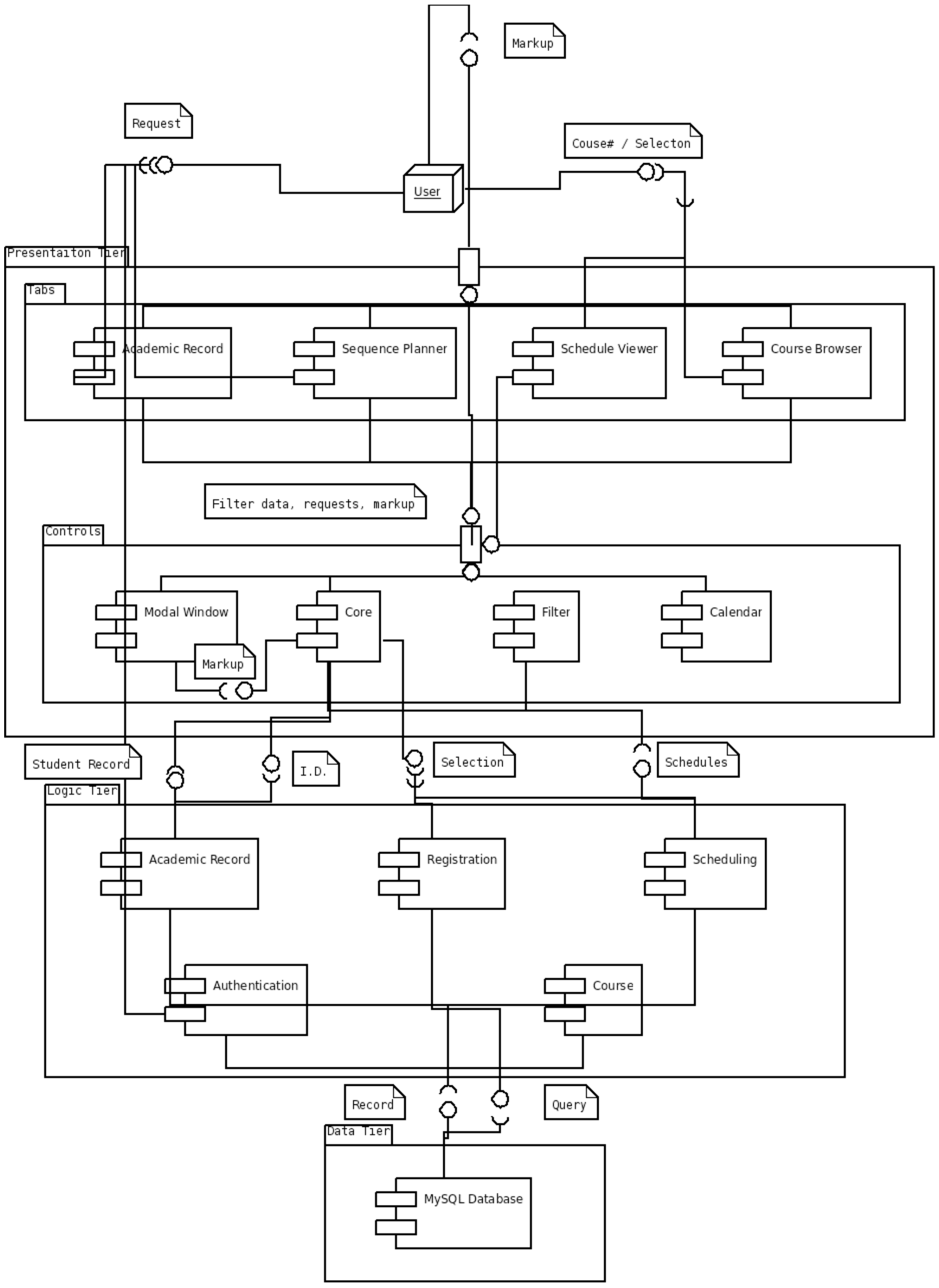


Figure 12.1 - UML Component Diagram

12.2 Brief Component Diagram Description

Figure 12.1 illustrates how the components of Planzilla are connected to each other.

12.2.1 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.

12.2.2 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.

12.2.3 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.

13 PRESENTATION TIER

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

13.1 Tabs

13.1.1 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.

13.1.2 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.

13.1.3 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.

13.1.4 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.

13.2 Controls

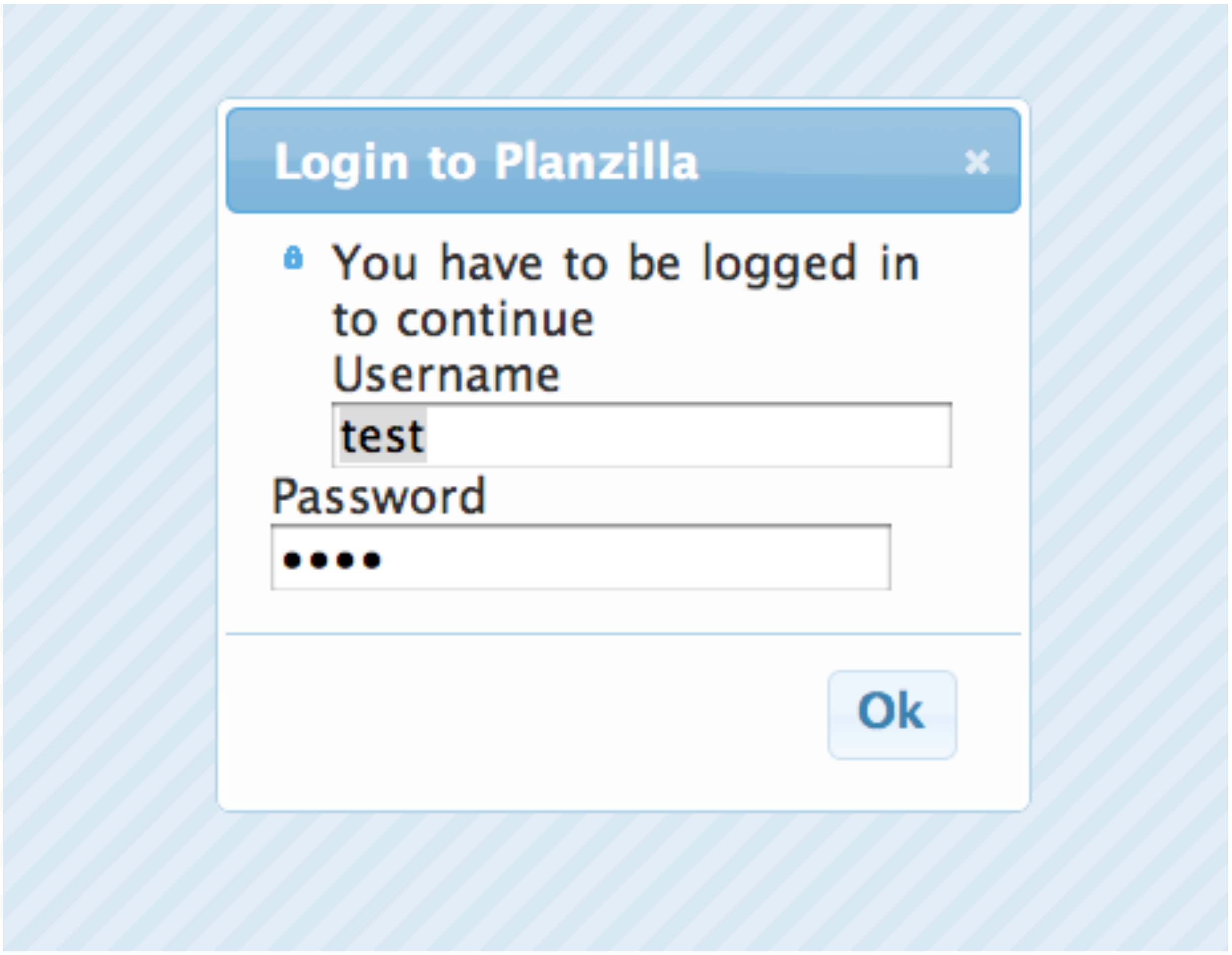
13.2.1 Core

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.

13.2.2 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 13.1.


Login to Planzilla

You have to be logged in to continue

Username

test

Password

••••

Figure 13.1 - Example of a modal window

13.2.3 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”.

13.2.4 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 13.2.

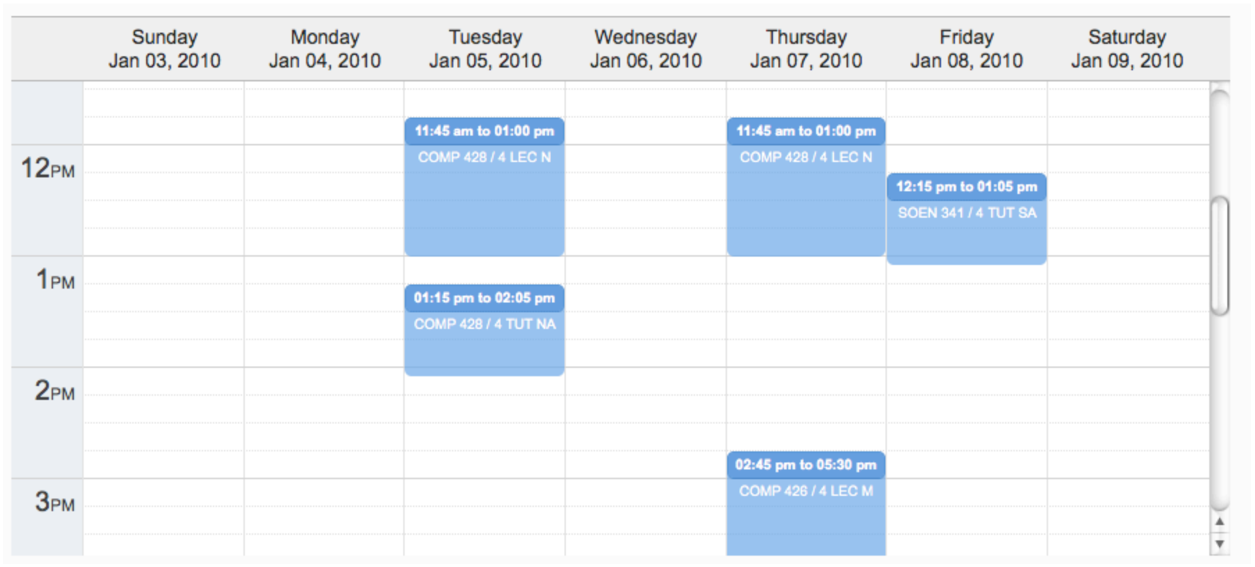


Figure 13.2 - Example of the calendar

14 LOGIC TIER

14.1 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).

14.2 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.

14.3 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.

14.4 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.

14.5 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.

15 DATA TIER

15.1 Overview

Figure 15.1 shows an overview of the database architecture designed for PlanZilla.

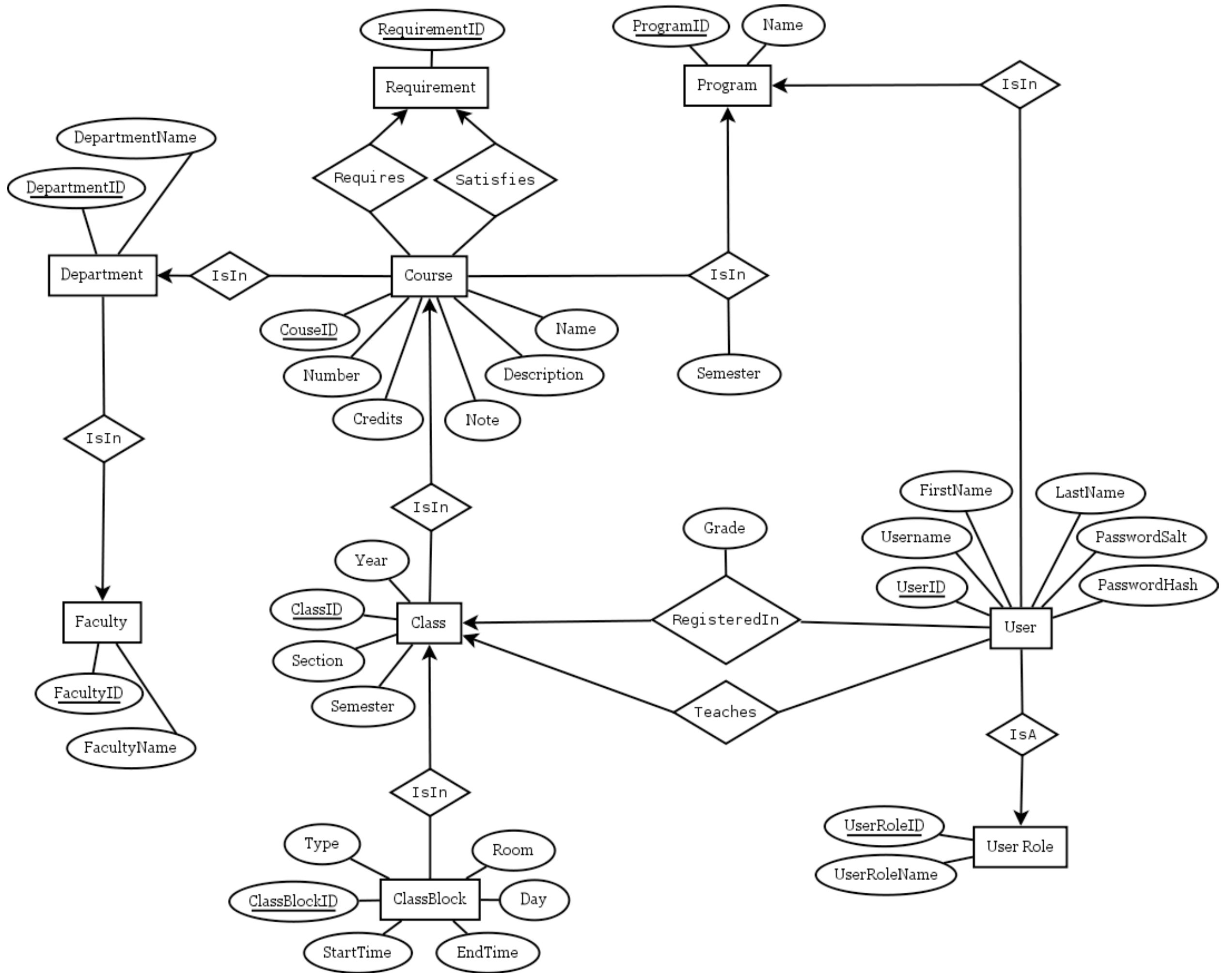


Figure 15.1 - Entity-Relationship Diagram

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

15.2 Users

Figure 15.2 shows how users credentials and information. Each user is assigned a role that determines what functions can be accessed, viewed, modified or removed.

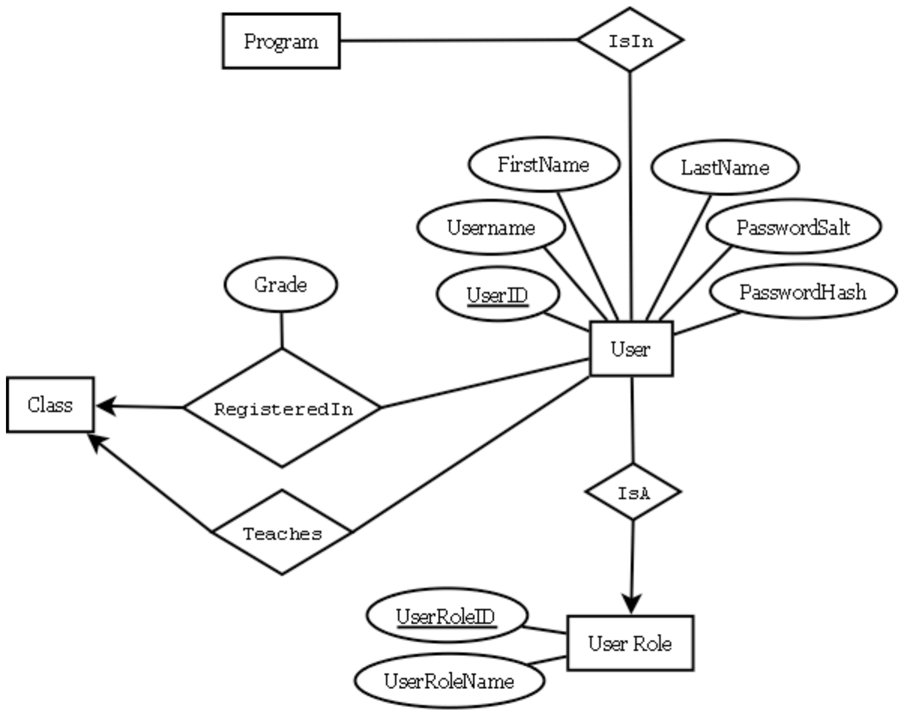


Figure 15.2 - User credentials and information.

15.2.1 Administrator

Users with the administrator role 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.

15.2.2 Student

Users with the student role are a special type of users that can be enrolled in a program and registered in courses. The RegisteredIn relationship is used for both present and past courses. The Grade attribute represents the current status of the course. If a course has been completed, the field will contain the achieved grade. If the course is currently being taken, then it will display “IP”. The grade field can also contain other statuses, such as “R” for classes to be repeated or “DISC” for discontinued courses.

15.2.3 Teacher

Users with the teacher role are a special type of users. They can teach different classes. The Teaches relationship is made with a specific class and not the course. This is because it is possible to have many teachers for one course. For instance, there may be more than one teacher giving different lecture sections and other teachers in charge of tutorials and labs.

15.3 Courses

Figure 15.3 focuses on how a course's information is listed in the database.

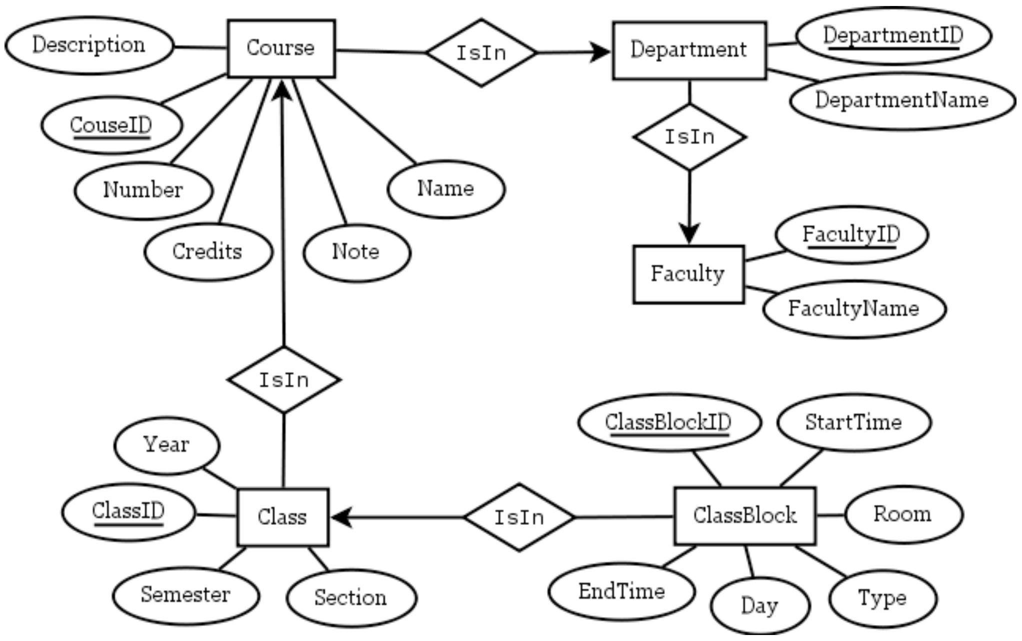


Figure 15.3 - Courses in the database

15.3.1 Course

A course is uniquely identified by its department and course number. For this reason, each course must be related to Department with the IsIn relationship. The IsIn relationship uses short four letter department names such as COMP or SOEN. The Name attribute of Course is simply the full name of the course and the Description attribute contains the full description of the course. The number of credits is also stored within the course. A Note attribute is contains special information for users with the administrator and teacher roles.

15.3.2 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 year, semester, section and type for a given class. The Section attribute is used to identify the class from other classes of the same course. The Type attribute is used to distinguish between lectures, tutorials and labs. Because of the large number of attributes required to uniquely identify a class, an extra attribute has been added: ClassID. The ClassID attribute is used internally by the database to relate users to classes.

15.3.3 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.

15.3.4 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.

15.4 Requirements

Figure 15.4 depicts how the prerequisites for a course are stored in the database.

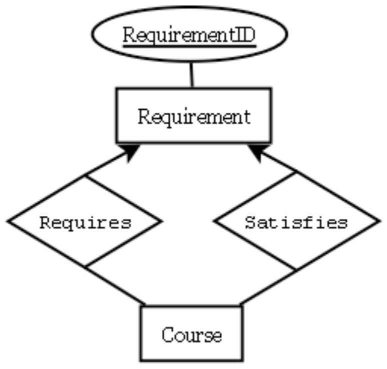


Figure 15.4 - Representation of requirements

15.4.1 "AND" Requirements

The simplest case of a requirement is a course which is dependent on one or more other courses. For example, let course A require courses B and C. In the database, course A would be related to two different requirements, R1 and R2, with the Requires relationship. One requirement, R1, would in turn be related to course B and R2 would be related to course C using the Satisfies relationship.

15.4.2 "OR" Requirements

The second requirement case is when a course is dependent on two or more courses that are equivalent. For example, let course A require course B or C. In the database, course A would be related to one requirement, R1, with the Requires relationship. The requirement R1 would in turn be related to both course B and C using the Satisfies relationship. This means that if the user has completed either course B, course C or both then course A’s requirement is satisfied.

15.4.3 Combination of "AND" and "OR" Requirements

There are more complex cases of requirements, in which groups of required courses are equivalent. This is only one example. Let course A require course B and course C or D. In the database, course A would be related to two different requirements, R1 and R2, with the Requires relationship. One requirement, R1, would be related to course B and R2 would be related to course C and D. This makes course A available only if the user has completed course B and course C or D (or both).

15.5 Programs

Figure 15.5 shows how a program is stored in the database.

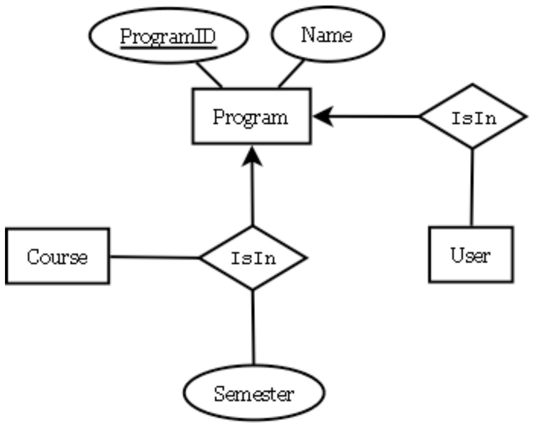


Figure 15.5 - Representation of a program

15.5.1 Program Options

A program can have many possible options. However, each option of a program corresponds to a different course sequence. For this reason, each program option has its own row in the Program entity and is uniquely defined by the ProgramID attribute and has its full name in the Name attribute. An example of a program would be Software Engineering.

15.5.2 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 attribute is present in the IsIn relationship between the Course and Program entities. The courses that have to be taken to meet the requirements of the sequence are represented using the established Requirement entities, as seen in section 15.4.

15.6 Final Database Model

Figure 15.6 shows the entity-relationship model that was reverse-engineered by the MySQL Workbench from the database built for PlanZilla.

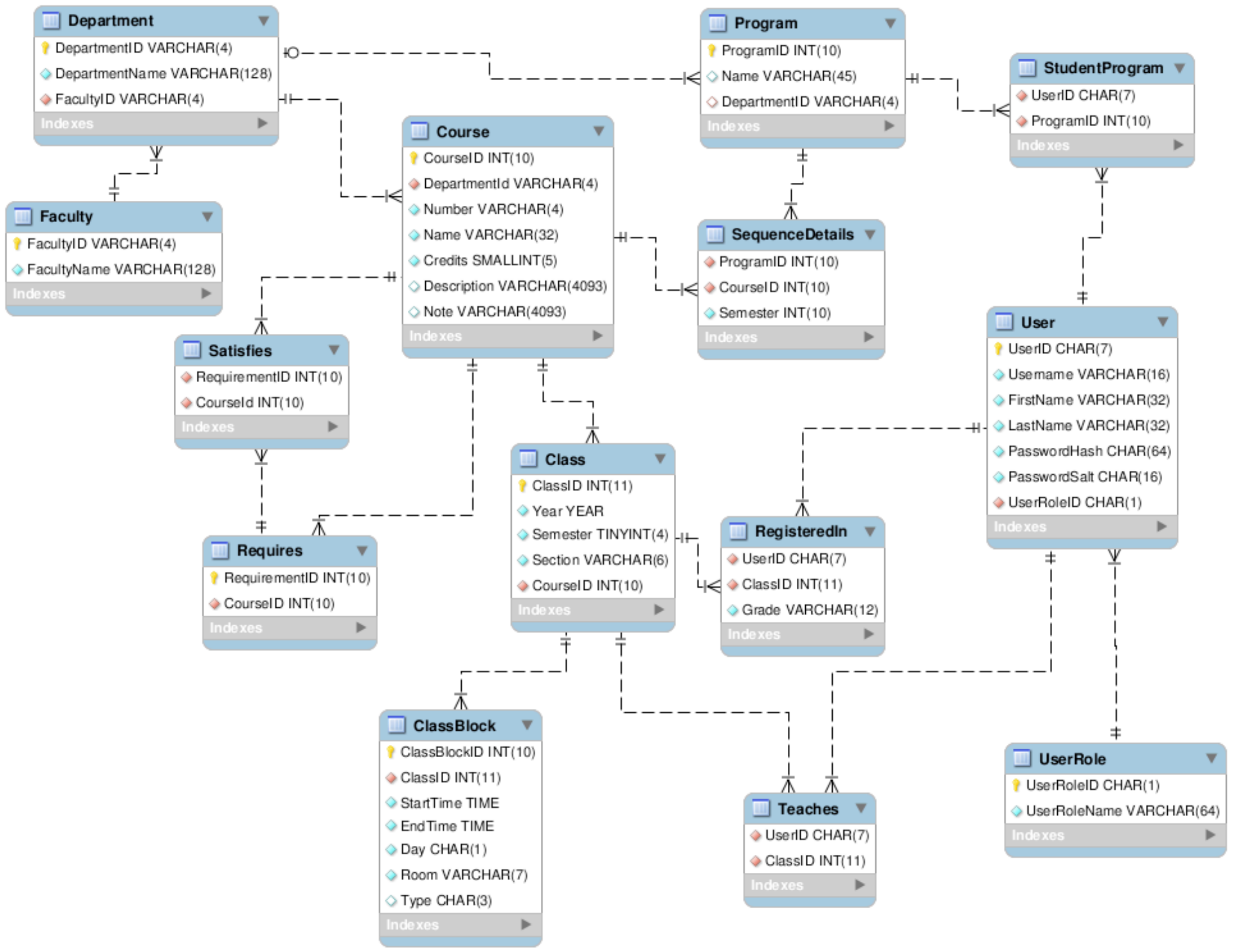


Figure 15.6 - Reverse engineered entity-relationship model for the PlanZilla database.

16 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.

16.1 Activity Diagram

Figure 16.1 shows a UML 2.0 activity diagram representing the process expressed by UC 11 (Generate Schedule Advanced).

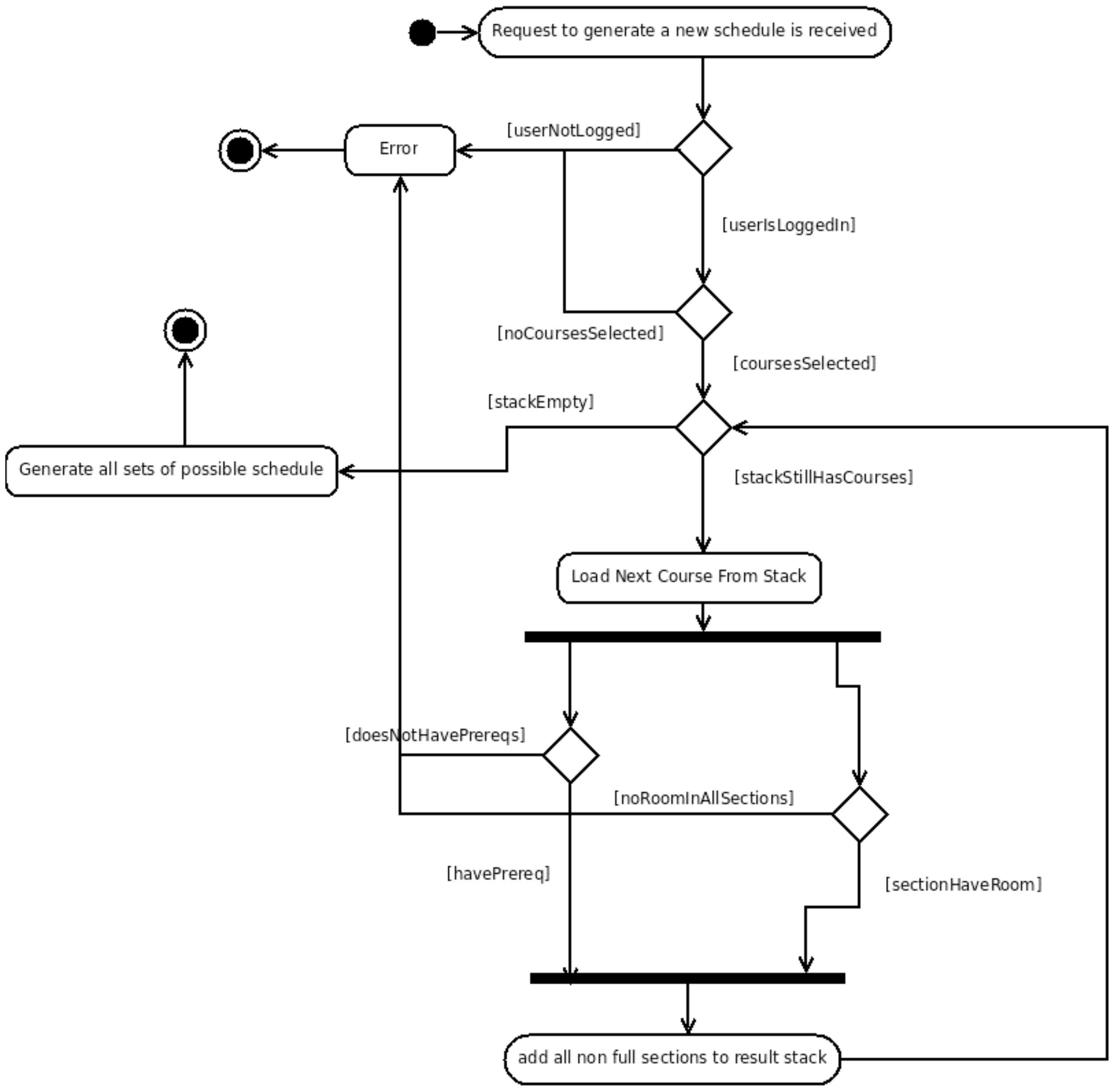


Figure 16.1 - Activity diagram

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.

17 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.

17.1 Deployment Diagram

Figure 17.1 shows the deployment diagram for the PlanZilla web application.

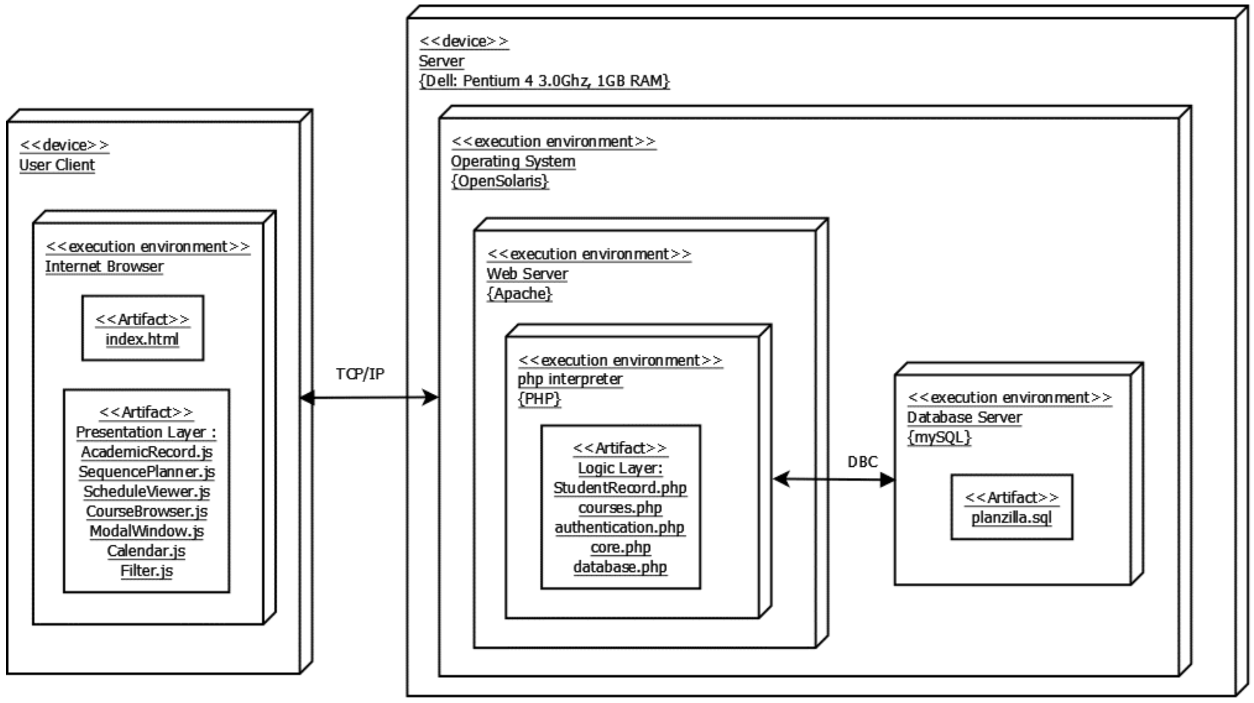


Figure 17.1- Deployment diagram.

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.

17.1.1 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.

17.1.1.1 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.

17.1.1.2 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.

17.1.1.3 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.

17.1.2 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.

18 LOGICAL VIEW

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

18.1 Class Diagram

Figure 18.1 shows the class diagram for the PlanZilla application.

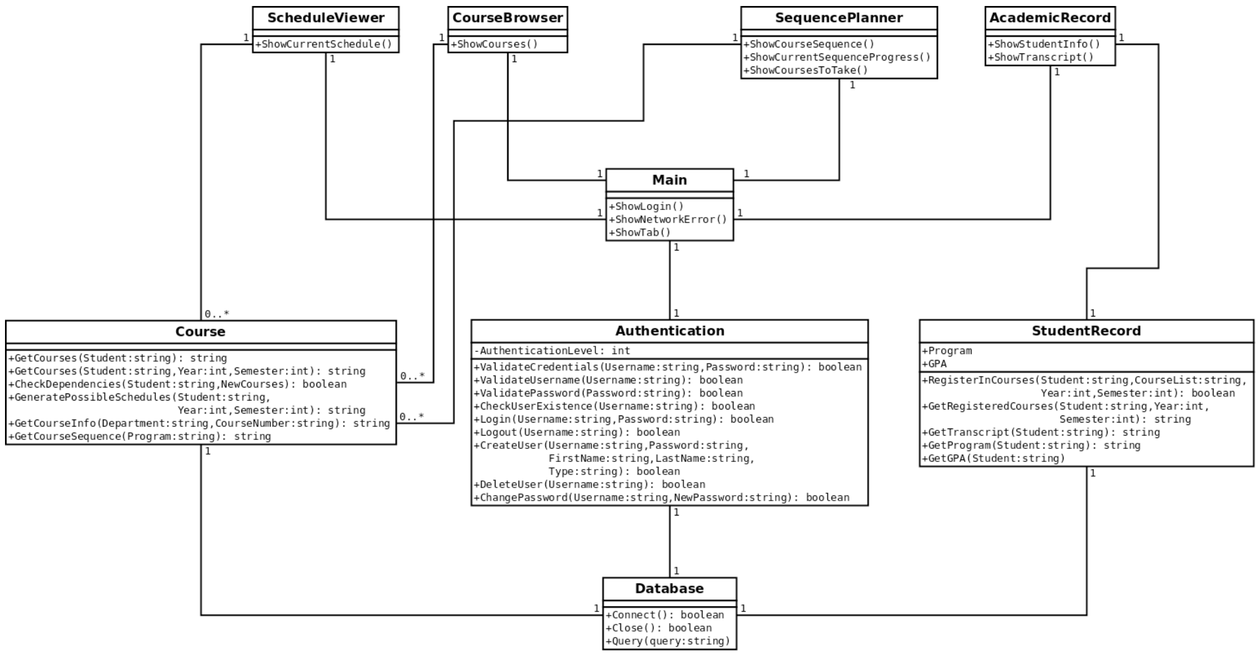

ScheduleViewer
+ShowCurrentSchedule()
CourseBrowser
+ShowCourses()
SequencePlanner
+ShowCourseSequence()
+ShowCurrentSequenceProgress()
+ShowCoursesToTake()
AcademicRecord
+ShowStudentInfo()
+ShowTranscript()
Main
+ShowLogin()
+ShowNetworkError()
+ShowTab()
Course
+GetCourses(Student:string): string
+GetCourses(Student:string,Year:int,Semester:int): string
+CheckDependencies(Student:string,NewCourses): boolean
+GeneratePossibleSchedules(Student:string, Year:int,Semester:int): string
+GetCourseInfo(Department:string,CourseNumber:string): string
+GetCourseSequence(Program:string): string
Authentication
-AuthenticationLevel: int
+ValidateCredentials(Username:string,Password:string): boolean
+ValidateUsername(Username:string): boolean
+ValidatePassword(Password:string): boolean
+CheckUserExistence(Username:string): boolean
+Login(Username:string,Password:string): boolean
+Logout(Username:string): boolean
+CreateUser(Username:string,Password:string, FirstName:string,LastName:string, Type:string): boolean
+DeleteUser(Username:string): boolean
+ChangePassword(Username:string,NewPassword:string): boolean
StudentRecord
+Program
+GPA
+RegisterInCourses(Student:string,CourseList:string, Year:int,Semester:int): boolean
+GetRegisteredCourses(Student:string,Year:int, Semester:int): string
+GetTranscript(Student:string): string
+GetProgram(Student:string): string
+GetGPA(Student:string)
Database
+Connect(): boolean
+Close(): boolean
+Query(query:string)
1
0..*

Figure 18.1 - Class diagram

18.1.1 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.

18.1.2 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.

18.1.3 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.

18.1.4 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.

18.1.5 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.

18.1.6 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.

18.1.7 Course

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

18.1.8 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.

18.1.9 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.

18.1.10 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.

19 MODULE INTERFACE SPECIFICATION

19.1 Authentication Class

19.1.1 Detailed Description

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

19.1.2 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)

19.1.3 Public Attributes

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

- **$AuthenticationLevel**
  User authentication level.

19.1.4 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

19.1.5 Member Data Documentation

Authentication:: $AuthenticationLevel

User authentication level:

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

19.1.6 Source File

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

- www/php/Authentication.php

19.2 Course Class

19.2.1 Detailed Description

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

19.2.2 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)

19.2.3 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.

19.2.4 Source File

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

- www/php/Course.php

19.3 Database Class

19.3.1 Detailed Description

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

19.3.2 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.

19.3.3 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.

19.3.4 Source File

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

- www/php/Database.php

19.4 StudentRecord Class

19.4.1 Detailed Description

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

19.4.2 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)

19.4.3 Private Attributes

The following are the private attributes for the StudentRecord class:

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

19.4.4 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.

19.4.5 Member Data Documentation

StudentRecord::$GPA [private]

Student's GPA

StudentRecord::$Program [private]
Student's Program

19.4.6 Source File

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

- www/php/StudentRecord.php

20 DYNAMIC DESIGN SCENARIOS

20.1 Browse Course List

20.1.1 System Sequence Diagram 1 (SSD1)

Figure 20.1 shows the system sequence diagram for the Browse Course List use case (UC6).

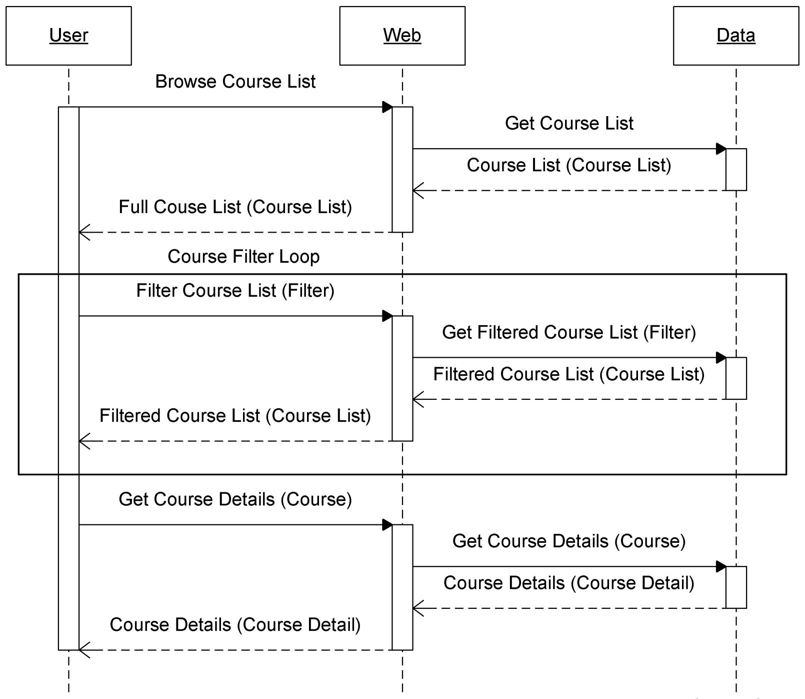

User
Web
Data
Browse Course List
Get Course List
Course List (Course List)
Full Couse List (Course List)
Course Filter Loop
Filter Course List (Filter)
Get Filtered Course List (Filter)
Filtered Course List (Course List)
Filtered Course List (Course List)
Get Course Details (Course)
Get Course Details (Course)
Course Details (Course Detail)
Course Details (Course Detail)

Figure 20.1 - 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.

20.1.2 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

20.2 Generate Schedule (Advanced)

20.2.1 System Sequence Diagram 2 (SSD2)

Figure 20.2 shows the system sequence diagram for the Generate Schedule (Advanced) use case (UC11).

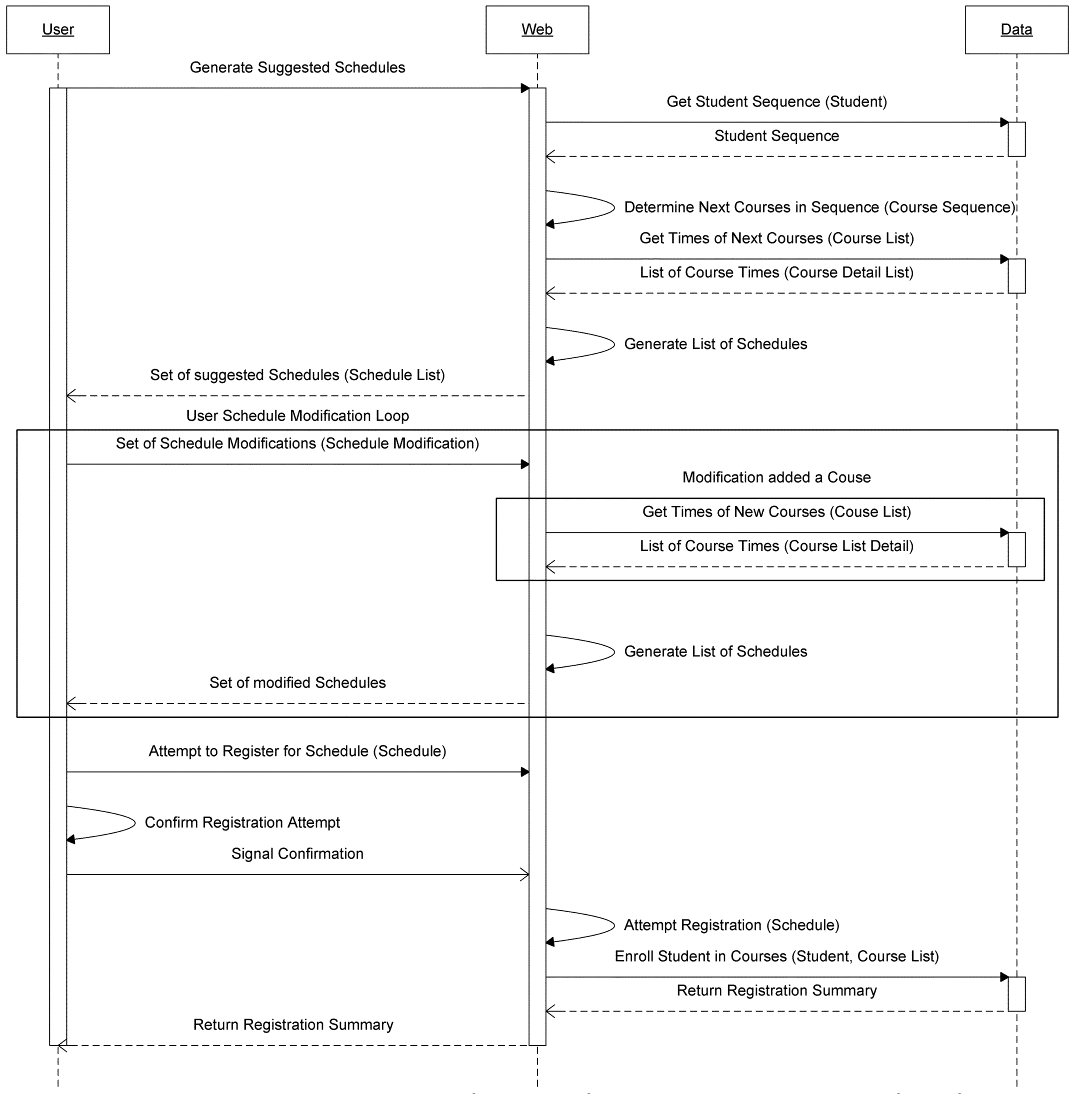

User
Web
Data
Generate Suggested Schedules
Get Student Sequence (Student)
Student Sequence
Determine Next Courses in Sequence (Course Sequence)
Get Times of Next Courses (Course List)
List of Course Times (Course Detail List)
Generate List of Schedules
Set of suggested Schedules (Schedule List)
User Schedule Modification Loop
Set of Schedule Modifications (Schedule Modification)
Modification added a Couse
Get Times of New Courses (Couse List)
List of Course Times (Course List Detail)
Generate List of Schedules
Set of modified Schedules
Attempt to Register for Schedule (Schedule)
Confirm Registration Attempt
Signal Confirmation
Attempt Registration (Schedule)
Enroll Student in Courses (Student, Course List)
Return Registration Summary
Return Registration Summary

Figure 20.2 - 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.

20.2.2 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 object associated with record data

20.2.3 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:	♦ Student object associated with GPA

20.2.4 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:	♦ JSON objects instanciated

20.2.5 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:	♦ Date objects are associated with the WeekCalendar instance.

20.2.6 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:	♦ Course objects instanciated

20.2.7 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:	♦ Course object's seatsFree attribute modified

~~20.2.8 Operational Contract 2.8 (CO2.8)~~ (Scoped out)

~~**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:**~~	~~♦ RegRequest object's valid attribute modified~~

~~20.2.9 Operational Contract 2.9 (CO2.9)~~ (Scoped out)

~~**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 object's registeredIn atribute modified~~

20.3 Miscellaneous

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

20.3.1 Operational Contract 3.1 (CO3.1)

Name:	**validateUsername()**
Related Use Case:	Login
Preconditions:	• User attempts to log on to the system
Post Conditions:	♦ Modified validID, validPW attributes in Authentication object

20.3.2 Operational Contract 3.2 (CO3.2)

Name:	**validatePwd ()**
Related Use Case:	Login
Preconditions:	• User attempts to log on to the system
Post Conditions:	♦ Modified validPW attribute in Authentication object

20.3.3 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 object instanciated

20.3.4 Operational Contract 3.4 (CO3.4)

Name:	**DBgetUsers()**
Related Use Case:	Login
Preconditions:	• Database table of users account data exists
Post Conditions:	♦ User objects are instanciated

20.3.5 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 Unique ID is instanciated

20.3.6 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:	♦ User object is associated with a unique ID

20.3.7 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 object is instanciated

20.3.8 Operational Contract 3.8 (CO3.8)

Name:	**displayError()**
Related Use Case:	ALL
Preconditions:	• An error is thrown
Post Conditions:	♦ Error Object instanciated

20.3.9 Operational Contract 3.9 (CO3.9)

Name:	**logEvent()**
Related Use Case:	ALL
Preconditions:	• A meaningful state change has occurred in the system
Post Conditions:	♦ An event object is instanciated

21 REVISION HISTORY

The following table contains all official released revisions of the Planzilla application meant for testing. For a full log of all revisions during development, go to the following website http://www.assembla.com/code/team7/git/changesets.

Table 21.1 - Revision History

Date	Version	Description	Author
04/04/2010	0.1	Early Pre-Alpha Release meant to try a first run through tests. Most functionality not present. Build failed testing phase as predicted.	Marc-André, Corey, Julia, Sebastien, Matt, Andreas, Eric, J.B.
06/04/2010	0.3	Added few core functionalities. Database now linked to front-end, displaying correct information. Build still fails during testing for missing functionalities.	Corey, Eric, Matt, Julia, Andreas
09/04/2010	0.5	Added more back-end logic and front-end mock-ups. Fixed bug with database. Build still fails during testing for missing functionalities.	Eric, Corey, Sebastien, Marc
10/04/2010	1.0	Major push to complete back-end logic and front-end. Linked both ends. Build fails for minor functionality issues.	Corey, Eric, Matt, Julia, Andreas, Sebastien, J.B., Marc
10/04/2010	1.4111	Minor bug fix. Build passes testing phases. Official release build.	Corey, Eric, Matt, Julia, Andreas, Sebastien, J.B., Marc

22 TESTING REPORT

This section contains all information on the testing of the Planzilla web application. The first section is an update on the current status of the testing such as what was tested and the results of these tests. The second section is a description of the performed tests along with all of their test cases.

22.1 Test coverage

22.1.1 Tested Items

Note: Latest test reports can be found in Appendix A and B towards the end of this document.

The application was tested for all listed requirements which mean the following group of functionalities were tested if they have met all requirements that were designed from requirements document (for more details on test cases, please consult section 3.2.2):

- Initial Start up (Req. Test cases 1.1.1 to 1.1.4)
  High priority. If start up does not work, nothing will work.

- Authentication (Req. Test cases 1.2.1 to 1.2.7)
  High priority. It is the basis of data security for user. Also, the application was designed to only enable certain functionalities once user is logged in.

- Schedule viewer (Req. Test cases 1.3.1 to 1.3.4)
  High priority. It is one of the main views for the user. Displays all information about current scheduling.

- Room/Professor Schedule (Req. Test cases 1.4.1 and 1.4.2)
  Lower priority, but only contains 2 test cases. Helps student choose their courses.

- Course Browsing (Req. Test cases 1.5.1 to 1.5.4)
  Medium priority. Helps students find courses and information about the courses.

- Program sequence (Req. Test cases 1.6.1 to 1.6.4)
  Medium priority. Helps student find what courses they should take.

- Academic Records (Req. Test cases 1.7.1 to 1.7.3)
  Low priority. Allows student to see information about his current and past courses.

- Activity log (Req. Test cases 1.8.1 to 1.8.3)

Low priority. Allows student to see past actions on authentication done from his account.

- Schedule Planner (Req. Test cases 1.9.1 to 1.9.12)
  High Priority. The key functionality of planzilla which will save student a lot of time when trying to find a schedule.

- Non-functional requirement (Req. Test cases 2.1 to 2.12)
  Medium Priority. Not related to functionality but all test cases are related to the quality of the application.

The result of these tests on the latest build can be seen from the latest Requirement Testing Checklist Report in Appendix A of this document.

Although, implementation is not yet complete, several modules or class have been tested due to their importance in the proper functioning of this application. These are:

- Database module
  This module contains all the information needed for the application to work properly such as course information and user information.

- Authentication class
  This is the basis of user security for this application. Functionalities such as schedule saving or registration can only be enabled when user is logged in.

- Course class
  This class treats all functionalities related to courses such as getting course information, getting course sequence and most important of all, generating course schedules.

- StudentRecord class
  This class displays information from student's transcript such as past courses and current program. This information is necessary for suggestion course schedules.

- Main class
  This is the core of the program which connects the front-end and the back-end.

22.1.2 Untested Items of Interest

All important items are actually being tested on each release build. However, smaller classes such as the Schedule viewer, Course Browser, Sequence Planner and Academic Record should also be tested. They do not contain any major logic, they are the layer in which the front-end communicates with; therefore, they are still should be tested for discrepancies in design.

22.2 Test Cases

Note: The numbering of test cases in this section has been set to correspond with the same numbering in their checklists.

22.2.1 Unit Testing

This section will describe the testing of the database and the authentication module. Both are essential parts of the application; therefore, they are highest in our unit testing priority. The testing is done using a black box approach: queries are sent to the modules, and a check is done on what is returned by the module. If the result is invalid, the test fails.

For the database, the tests consist of checking if it is able to handle the storing and retrieving of information. As for the authentication module, the tests consist of checking if the module can manage user credentials in a secure manner. More precisely, users need to be able to login, logout and manage user accounts without and issues or insecurities.

The following consists of all the test cases for the *database unit testing*:

1. **Design requirement**

   1.1. *Database handles storing course information such as the department, course number, credits, course description, different classes for the same course with a time and a teacher.*

   Check if there are tables matching the ER diagrams from the software design document. Using phpmyadmin, it can be verified that there are tables for departments, courses, classes and teachers. The logical links between the tables are also present, with referential integrity enforced in the database. In this case, this means that foreign keys are used between the course and department tables (a course needs to be in a department), between the classes and courses (you can't have a class without a course).

   1.2. *Database is able to store student and teacher information.*

   There are different ways to test this part of the database as described in the higher-level ER diagram. The test was done using a single user table, with a field indicating the type of the user. The user table can store usernames, passwords, and personal information associated with the student or teacher. The test was just to check that these fields were present as indicated in the ER diagram.

   1.3. *Database is able to represent teachers teaching different classes and students being registered in different classes.*

Check if a table representing the relationship "teachers" between a teacher and a class is present, effectively enabling the representation of teachers teaching more than one class. Also, check if a table representing the relationship "RegisteredIn" is present, enabling a student to be registered in more than one class.

2. Implementation Tests

2.1. Obtaining course information

Using phpmyadmin, the contents of the course table could be listed, which contain all the course information. As the database is being tested, this is a sufficient test to know that the database can be used to obtain course information. If it can't be done in the database, then it can't be done in the front-end either.

2.2. Obtaining student information

Using phpmyadmin, the contents of the student table could be listed, which contain all student information. A simple mysql query can be built to select a student by its student ID, so that information for a single student is obtained.

3. Other Tests

3.1. SQLInjectMe does not detect any SQL injection points.

SQLInjectMe is a firefox plugin that checks for html forms in a website, and tries common SQL injection attempts agains the various fields of the html forms. It then gives you a summary of the attempted SQL injection attacks, with those that were successful and those that were unsuccessful. For PlanZilla, SQLInjectMe reported no SQL injection success.

3.2. Nikto does not find any relevant security issues in the web application.

Nikto is a popular web application vulnerability scanner. It works by using a collection of scripts that try to detect all sorts of security flaws in web applications. It then gives you a complete list of anything it could find, usually including a vulnerability ID so that further information can be found (known vulnerabilities are indexed into vulnerability databases, where information is gathered about them, making it easier for security experts to refer to a particular vulnerability and avoid duplication). Usually nikto will also output a lot of false positives, or comments about particularities of the web application that are not necessarily security issues. For instance, if it finds a file called "admin.html", it will output a comment saying that "admin.html" may be a worth a closer look, but it does not imply that vulnerabilities were found in it. For PlanZilla, nikto did not reveal any relevant security issues.

The *authentication module test* both uses white box and black box approaches for testing. The white box approach is used to check if all requirements from design are present in the module's code. As to check if the functionalities work and are secure, the black box approach is used by giving an input to the module and check if the module correctly performs the necessary tasks or returns the correct output.

This module is the basis of security in this application since it makes sure that unauthorized users are not able to access and use modules in ways that are not permitted to them. To thoroughly accomplish this, the module needs to be tested for security flaws that can potentially lead to system and data security issues. Google offers such a web application called Skipfish that does exactly this (For more information, visit http://code.google.com/p/skipfish/).

Google created Skipfish to smoke test their web applications before deployment. This tool has been written by the company's best security engineers. In March 2010, Google decided to make this tool public. By using this, Planzilla can be thoroughly and effectively tested, since all the security flaws that we know of, are being taken into account by this tool. Furthermore, Skipfish was chosen over other publicly available tools, such as Nikto and Nessus, because of following advantages:

- High performance (can achieve 500+ requests/seconds with a very modest CPU, network, and memory footprint)
- Multiplexing single-thread, fully asynchronous network I/O and data processing model that eliminates memory management, scheduling, and IPC inefficiencies present in some multi-threaded clients.
- Smart response caching and advanced server behaviour heuristics are used to minimize unnecessary traffic.
- Smart, up-to-date and active (but hopefully non-disruptive) security checks (Google engineers update this tool almost every day!)

The following list contains all the tests concerning the authentication module including a simplified version of Skipfish's numerous tests:

1. Design requirement

1.1. The Authentication module can communicate with the Main module and the MySQL database without any issues.

1.2. The Authentication module forces clients to have proper access rights for each module by requesting them to login using their credentials.

The following tests only checks if the member functions are present in the code. This is to be manually tested by the developer.

1.3. ValidateCredentials is implemented.
1.4. ValidateUsername is implemented.
1.5. ValidatePassword is implemented.
1.6. CheckUserExistence is implemented.
1.7. Login is implemented.
1.8. Logout is implemented.
1.9. CreateUser is implemented.
1.10. DeleteUser is implemented.
1.11. ChangePassword is implemented.
1.12. $AuthenticationLevel is an implemented as an attribute.

2. Implementation Test

For the following test, please refer the design documents for their functionalities. Test if they do exactly what they were designed to do. These should also to be tested by the developer only.

2.1. ValidateCredentials works as intended by design.
2.2. ValidateUsername works as intended by design.
2.3. ValidatePassword works as intended by design.
2.4. CheckUserExistence works as intended by design.
2.5. Login works as intended by design.
2.6. Logout works as intended by design.
2.7. CreateUser works as intended by design.
2.8. DeleteUser works as intended by design.
2.9. ChangePassword works as intended by design.
2.10. $AuthenticationLevel works as intended by design.

For the following three section, these test cases are automated test by Google's Skipfish web application. Therefore, details about these tests will not be documented here. For more information about these tests please refer to Skipfish's website: http://code.google.com/p/skipfish/.

3. SkipFish High Risk Security Tests

3.1. Server-side SQL injection (including blind vectors, numerical parameters)
3.2. Explicit SQL-like syntax in GET or POST parameters.
3.3. Server-side shell command injection (including blind vectors).
3.4. Server-side XML / XPath injection (including blind vectors).
3.5. Format string vulnerabilities.
3.6. Integer overflow vulnerabilities.
3.7. Locations accepting HTTP PUT

4. SkipFish Medium Risk Security Tests

4.1. *Stored and reflected XSS vectors in document body (minimal JS XSS support present).*
4.2. *Stored and reflected XSS vectors via HTTP redirects.*
4.3. *Stored and reflected XSS vectors via HTTP header splitting.*
4.4. *Directory traversal (including constrained vectors).*
4.5. *Assorted file POIs (server-side sources, configs, etc).*
4.6. *Attacker-supplied script and CSS inclusion vectors (stored and reflected).*
4.7. *External untrusted script and CSS inclusion vectors.*
4.8. *Mixed content problems on script and CSS resources (optional).*
4.9. *Incorrect or missing MIME types on renderables.*
4.10. *Generic MIME types on renderables.*
4.11. *Incorrect or missing charsets on renderables.*
4.12. *Conflicting MIME / charset info on renderables.*
4.13. *Bad caching directives on cookie setting responses.*

5. SkipFish Low Risk Security Tests

5.1. *Directory listing bypass vectors.*
5.2. *Redirection to attacker-supplied URLs (stored and reflected).*
5.3. *Attacker-supplied embedded content (stored and reflected).*
5.4. *External untrusted embedded content.*
5.5. *Mixed content on non-scriptable subresources (optional).*
5.6. *HTTP credentials in URLs.*
5.7. *HTML forms with no XSRF protection.*
5.8. *Bad caching directives on less sensitive content.*

22.2.2 Requirement Testing

This section contains all test cases that were derived from the requirements identified during the design phase. The tests are divided into two subsections. One for functional requirements which are derived directly use cases from the design document. The second is for non-functional requirements which are derived from the Quality Standards section of the same document.

1. Functional Requirement Test Cases

1.1. Initial Start Up

1.1.1. *User is able to launch the web application using a web browser entering this address: https://team7.ath.cx/.*
Use a HTML 4.01 compliant browser or higher with JavaScript support.

~~*1.1.2. If a non-optimal browser was used to access this application, a warning message will appear but will let user continue.*~~ *(Scoped out)*
~~Use a browser that uses HTML 4.00 or lower with JavaScript support.~~

~~*1.1.3. If a non-supported browser was used to access this application, this application will display an error and will not launch.*~~ *(Scoped out)*
~~Use a browser that uses HTML 4.00 or lower without JavaScript support.~~

1.1.4. The application launches successfully and displays the correct initial screen: "View schedule".

1.2. Authentication

1.2.1. User is able to access a login page or module.
A login module appears at any point in the application.

1.2.2. User can enter credentials for login.
Use the following username/password: dummy/0mgh4x

1.2.3. If credential is valid, the user will be successfully logged in.
A status comment in the bottom of the screen indicates user login status.

1.2.4. If an invalid credential was used, the system will display an error message and ask user to try again.
Use the following usernames/passwords:
dumm/abc → Gives invalid username error.
notadummy/abc → Gives invalid password error.
notadummy/abc123 → Gives invalid username or password error.

1.2.5. While being logged in, the users can logout of the application.
Logout button is present and functional while logged in.

1.2.6. Accessing "Log out" successfully logs the user out of his account.
Application returns to login screen.

1.2.7. After having successfully logged out, the screen no longer display any information related to user's session or account and is taken back to default page.

1.3. Schedule Viewer

1.3.1. View Schedule button is available on the navigation bar.

1.3.2. View Schedule page is the default page after logged in.

1.3.3. A schedule is displayed on screen.

~~1.3.4. The schedule viewer correctly displays the current saved schedule in user's account. If no schedule has been set, that should also be reflected correctly on the viewer.~~ (Scoped out registering functionality)

1.4. ~~Room / Professor Schedule~~ (Scoped out)

~~1.4.1. From the Schedule Viewer, user can access specific room and professor schedules.~~

~~1.4.2. The Schedule viewer displays the schedule correctly.~~

1.5. Course Browsing

1.5.1. Browse Course List is available on the navigation bar.

1.5.2. Clicking on Browse Course List button will display the correct page.

1.5.3. From this page, user can select a department and course number from a list.

1.5.4. *Upon selecting a course, the correct information for that course is displayed.* Browsing through courses does not yield any errors.

1.6. Program Sequence

1.6.1. Program Sequence button is available from the navigation bar.

1.6.2. Clicking on Program Sequence will display the correct page.

1.6.3. A complete list of user's program course sequence is displayed correctly.

1.6.4. If the user is not enrolled in a program, an warning should be displayed to inform user that a sequence is not available.

1.7. Academic Record

1.7.1. Academic Record is available from the navigation bar.

1.7.2. Clicking on Academic Record will display the correct page.

1.7.3. *User's academic record page correctly displays previously and currently taken course and displays their grade correctly.*

~~1.8. Activity Log~~ (Scoped out)

~~*1.8.1. Activity Log is available from the navigation bar.*~~

~~*1.8.2. Clicking on Activity Log will display the correct page.*~~

~~*1.8.3. User's academic record correctly displays logged transactions from user's activity in the system.*~~

1.9. Schedule Planner

1.9.1. Schedule planner is available from the navigation bar.

1.9.2. Clicking on Schedule planner will display the correct page.

1.9.3. The page contains a Generate Schedule button.

1.9.4. Pressing this button will offer user a list of possible schedules.

1.9.5. Users are able to cycle through different schedule and choose a specific one.

1.9.6. The application will prompt for confirmation when a schedule is selected.

1.9.7. Once user has confirmed, the system confirms that the operation was successful.

1.9.8. Schedules are displayed correctly on the viewer.

1.9.9. While user is selecting schedule, the application offers the option to add time constraint or preferences.

1.9.10. Any changes in the constraints are reflected on the schedule viewer.

~~*1.9.11. Once a schedule is successfully selected. All changes and scheduling are saved and will be displayed the next time user logs in.*~~ *(Scoped out)*

1.9.12. *An error is generated if application cannot find a suitable schedule for the user according to his constraints.*

2. Non-Functional Requirement test cases

2.1. *The application responds quickly. Any regular requests should take less than 2 seconds except for schedule generation.* (Req. Section 3.3.1)
Check several functionalities such as clicking through navigation bar, login, logout etc.

2.2. *Generating schedules should not take more than 5 seconds.* (Req. Section 3.3.1)
Use the Generate schedule button and measure time of response.

2.3. *The system is user-friendly and intuitive. User should never be lost while navigating through application and can easily find sections.* (Req. Section 3.3.2)

~~2.4. *Each page contains a back button which takes user to the last visited page.* (Req. Section 3.3.2)~~ (Scoped out)

2.5. *The system should easily scalable.* (Can only be tested by certified a developer. Pass by default if build is released for testing unless indicated otherwise by test lead) (Req. Section 3.3.3)

2.6. *System should support a minimum of 5000 concurrent users.* (See Stress testing results) (Req. Section 3.3.3)

2.7. *If system cannot log user in due to large traffic, the application should return an error message to try again later.* (Req. Section 3.3.3)

2.8. *System is secure against security threats* (see Security Testing results). (Req. Section 3.3.4)

2.9. *User with disabilities should be able to use the application with accessibility features without any problems*. (Req. Section 3.3.5)
Run am accessibility application such as the Magnifier or Text-To-Speech. Check if these applications run normally when navigating through Planzilla.

2.10. *The application is printer friendly. Printing any page should not yield unwanted artefacts or extra white pages.* (Req. Section 3.3.6)
After having generated a schedule, print the page by using default browser printing function under file menu.

2.11. *User is able to resume a session without losing input data after a wireless connection is lost.* (Req. Section 3.3.7)
Navigate through application using a wireless connection. Disconnect and reconnect wireless and check if information has been lost.

2.12. Application is free from any graphical issues.

22.2.3 Stress testing

There are several moments during the year where the number of users can peak. This is during registration periods. Thousands of students are logged in at the same time to consult course information, register in their courses, or even drop them as they change their mind. Students often will constantly try to change schedule to match their needs. They will constantly check the availability for class availability changes and therefore, they will be using the application several times before they are satisfied. This kind of situation involves the following operations being stressed in particular: browsing courses, obtaining available classes for a course, registering in courses, generating possible schedules, etc. In other words, all functionalities of the system will be heavily in demand.

These situations can be tested by setting up a server in a controlled environment, in LAN with 4 other computers. Each computer would have gigabit network cards, and would be connected together with a gigabit router. The goal here is to reduce network latency to almost nothing. This setup will allow each computer, simulating a heavy load of students, to not lag down while trying to flood the server with hundreds of thousands of queries.

The test will be performed using a PERL script that simulates users using the system, through the use of libcurl (a library for various web operations that are normally done by a browser). The script will be doing a complete set of operations effectively simulating all sorts of demands that can be done with the web application. The same routine would be done over and over again, with information about the response time for each operation being stored for analysis of the system performance under stress conditions. Also, queries that are faking user interaction won't always be the same (for instance, always querying for the same user). This is to avoid possible caching from the server that would enable it to react better than it would have had in a real life situation. A single PERL script ran on 4 computers would not suffice to take the system down, so the script would be forking the simulation routine somewhere about 10 000 times and running it concurrently on each test computer. Hopefully, that should generate enough stress on the server in order to get valuable results from the stress test.

23 SYSTEM DELIVERY

This section contains a manual that describe how to properly install Planzilla on a web server and a second manual that informs users how to use the application.

23.1 Install Manual

23.1.1 Installation Instructions

Planzilla uses some server software and needs to be configured manually prior to deployment. Before attempting installation, please be sure you have permission to perform all the required tasks and any pertinent information - such as database credentials - to avoid an incomplete or failed installation.

23.1.2 Server Verification & Installation

Planzilla requires the use of the Apache Web Server with support for SSL and PHP 5 as well as MySQL. If you are installing Planzilla on its own machine, you may use a compatible AMP stack – such as XAMPP - or install the software separately. If you have an existing server for deployment, please make sure it is running at least PHP version 5 with support for MySQL. Otherwise you can upgrade to the latest version of PHP. Perl and Python are also required to execute the data mining and database population scripts.

NOTE: Installation of these software packages is beyond the scope of this document. Please refer to the software vendor's homepage for up-to-date versions of the software and installation instructions.

23.1.3 PlanZilla Installation

All the files required for PlanZilla are located in the *www* directory of the installation bundle. Be sure to extract the contents of the bundle to the appropriate directory of the web server. Also make sure that the directory has permission to execute PHP scripts.

23.1.4 PlanZilla Configuration

You can configure Planzilla by filling in the required values in the Config.php file. An example file named Config.php.example is provided for your convenience and is located in that same directory.

23.1.5 Database Preparation

Against the database, run the setup script *dbsetup.sql* found in the scripts directory of the installation bundle. One way to go about doing this would be to execute the command "mysql <db> -u <username> -p < scripts/dbsetup.sql" where '<db>' is the name of the mysql database and '<username>' is a valid user with sufficient privileges to modify this database.

23.1.6 Database Population

The purpose of this section is to populate the database with all the required information such as Courses, Course Sections, and Program Sequences. An example script, which will automatically pull data from the Concordia University course listing, is available under the scripts directory of the installation bundle. The script is called *generateAll.py*. When it is executed, it will generate a file called *INITIAL_DB.sql*, which is to be loaded manually, as in the previous step.

23.1.7 Deployment

The only way to add a user is through the database. In order to start adding students, the database has been set up with a default user “admin” and a default password “default”.

Congratulations, you have successfully set up Planzilla!

23.1.8 Software Downloads

URIs to the software used throughout the installation process and downloads of the required software can be found at the following homepages.

XAMPP - `http://www.apachefriends.org/en/xampp.html`

Apache Web Server - `http://httpd.apache.org/`

PHP - `http://php.net/`

MySQL - `http://www.mysql.com/`

Perl - `http://www.perl.org/`

Python - `http://www.python.org/`

23.2 Instruction Manual

Welcome to Planzilla, this manual will walk you through all of Planzilla features to make your day as a student as easy as it can be. The following is a description of how to use each feature of the PlanZilla web application.

23.2.1 Accessing Planzilla

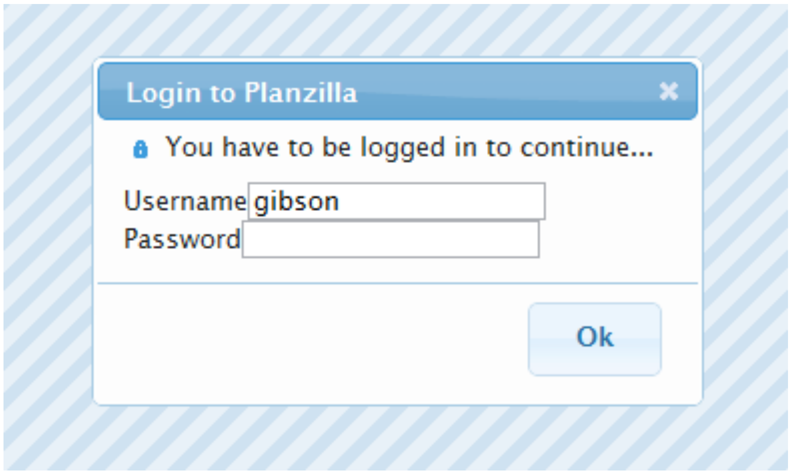
In order to access the PlanZilla web application, the following must be done:

1) Open a compatible web browser*.
2) Type this address into the url bar: https://team7.ath.cx/.

***NOTE**: This application is guaranteed to work with Firefox 3.5.8. Other web browsers or previous versions might be subject to unknown errors.

23.2.2 Login/Logout

If you are a registered user, you have a username and password. When prompted to login, a window will pop up as seen in Figure 23.1.



You have to be logged in to continue...

Username gibson

Password

Figure 23.1 - Login Screen

Enter your username and password where required. Then either click on the ‘OK’ button or press enter to login. If this is successful, the window should disappear. If not you will be prompted to try again.

23.2.3 Navigation Bar

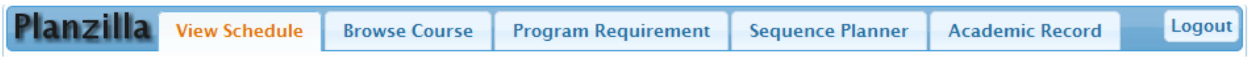
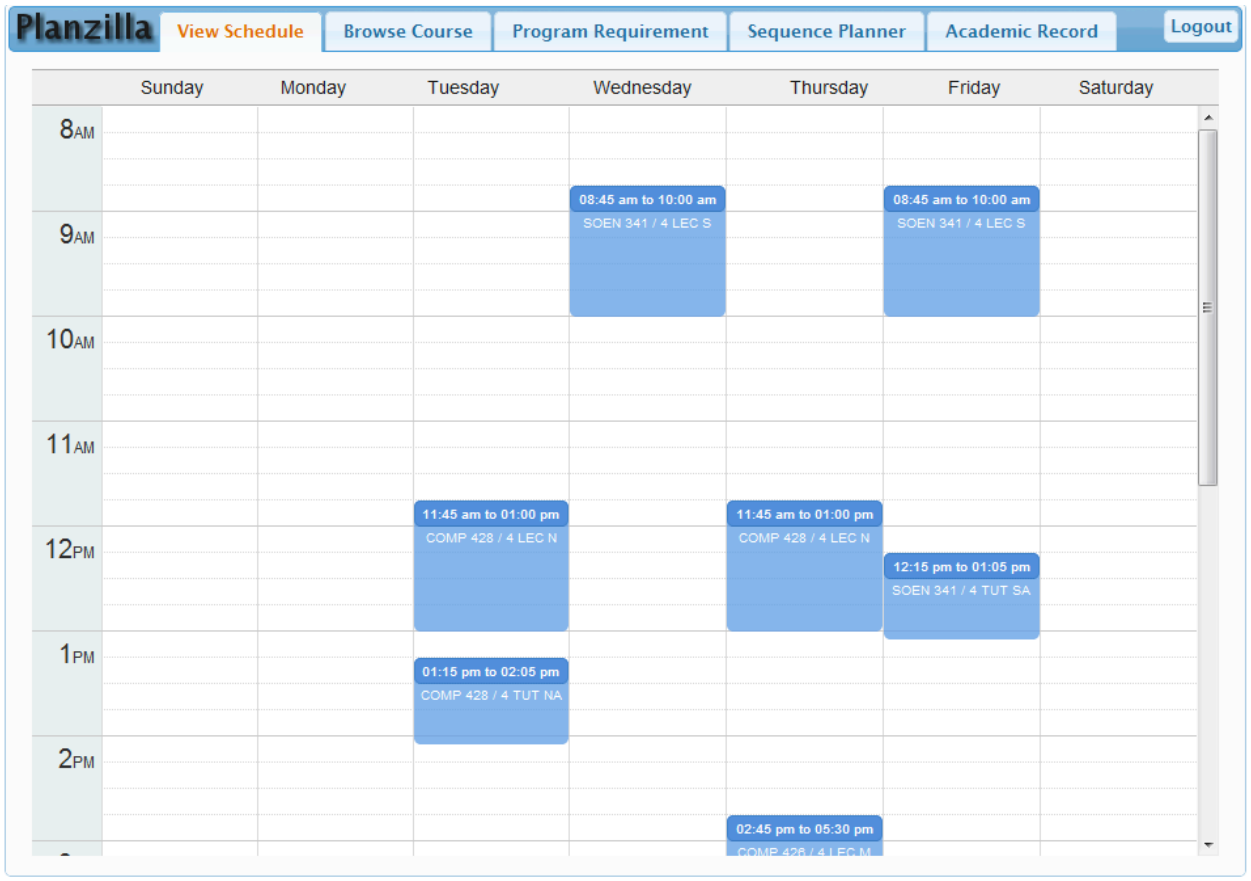


Figure 23.2 - PlanZilla navigation bar

The navigation bar consists of five tabs, as seen in Figure 23.2. Each tab contains its own function. By clicking on a tab, access to that specific function is gained. If the user is logged in, the logout button will also be displayed in the navigation bar.

23.2.4 View Schedule

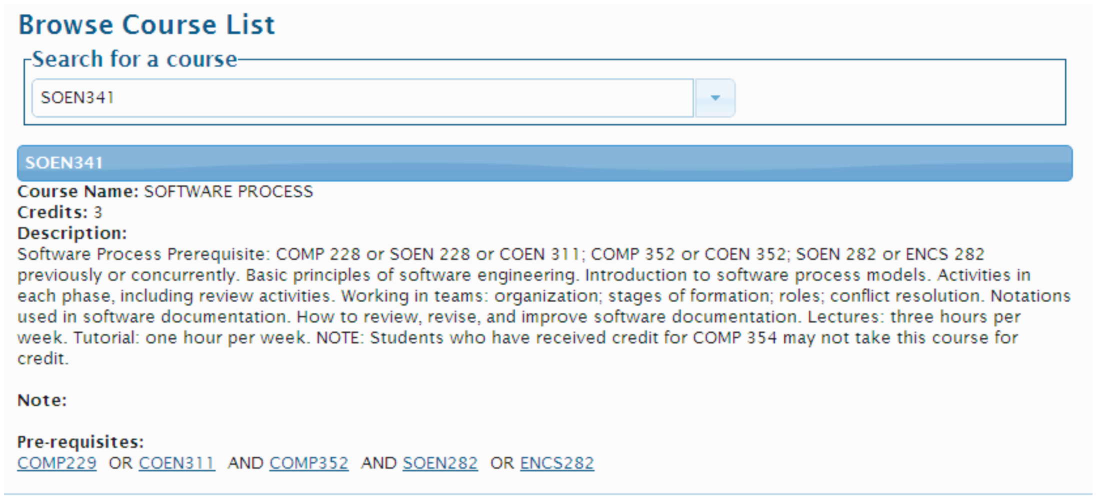


08:45 am to 10:00 am SOEN 341 / 4 LEC S
08:45 am to 10:00 am SOEN 341 / 4 LEC S
11:45 am to 01:00 pm COMP 428 / 4 LEC N
11:45 am to 01:00 pm COMP 428 / 4 LEC N
12:15 pm to 01:05 pm SOEN 341 / 4 TUT SA
01:15 pm to 02:05 pm COMP 428 / 4 TUT NA
02:45 pm to 05:30 pm COMP 426 / 4 LEC M

Figure 23.3 - View schedule screenshot

When the View Schedule tab is clicked, the user gains access to the currently registered schedule as seen in Figure 23.3.

23.2.5 Browse Course



Browse Course List

Search for a course

SOEN341

SOEN341

Course Name: SOFTWARE PROCESS
Credits: 3
Description:
Software Process Prerequisite: COMP 228 or SOEN 228 or COEN 311; COMP 352 or COEN 352; SOEN 282 or ENCS 282 previously or concurrently. Basic principles of software engineering. Introduction to software process models. Activities in each phase, including review activities. Working in teams: organization; stages of formation; roles; conflict resolution. Notations used in software documentation. How to review, revise, and improve software documentation. Lectures: three hours per week. Tutorial: one hour per week. NOTE: Students who have received credit for COMP 354 may not take this course for credit.

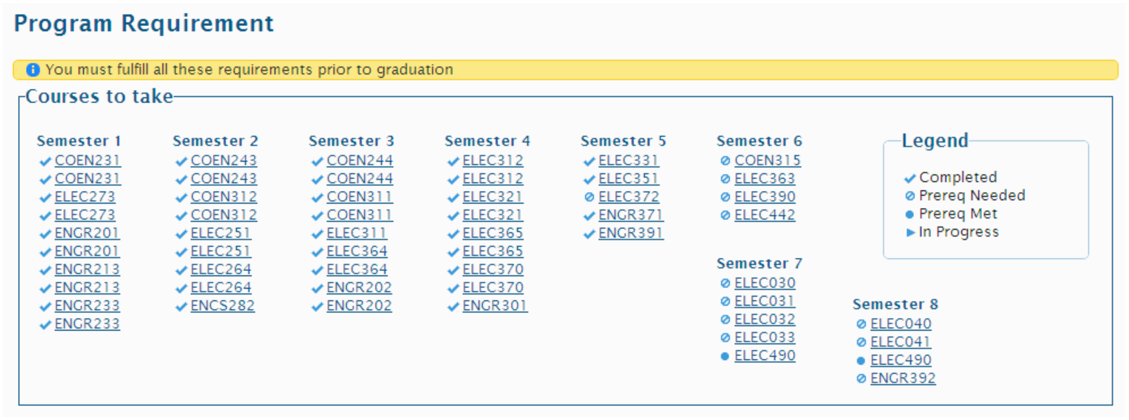
Note:

Pre-requisites:
COMP229 OR COEN311 AND COMP352 AND SOEN282 OR ENCS282

Figure 23.4 - Browse course screenshot

When the Browse Course tab is clicked, the user gains access to all course information. To view information for a specific course, select the course you wish to view from the drop down box. The course information will be loaded. To view another course within the same department, simply type in the new course name or code. A sample can be seen in Figure 23.4.

23.2.6 Program Requirement



Program Requirement

You must fulfill all these requirements prior to graduation

Courses to take

Semester 1	Semester 2	Semester 3	Semester 4	Semester 5	Semester 6	Semester 7	Semester 8
✓ COEN231	✓ COEN243	✓ COEN244	✓ ELEC312	✓ ELEC331	⊘ COEN315	⊘ ELEC030	⊘ ELEC040
✓ COEN231	✓ COEN243	✓ COEN244	✓ ELEC312	✓ ELEC351	⊘ ELEC363	⊘ ELEC031	⊘ ELEC041
✓ ELEC273	✓ COEN312	✓ COEN311	✓ ELEC321	⊘ ELEC372	⊘ ELEC390	⊘ ELEC032	● ELEC490
✓ ELEC273	✓ COEN312	✓ COEN311	✓ ELEC321	✓ ENGR371	⊘ ELEC442	⊘ ELEC033	⊘ ENGR392
✓ ENGR201	✓ ELEC251	✓ ELEC311	✓ ELEC365	✓ ENGR391		● ELEC490	
✓ ENGR201	✓ ELEC251	✓ ELEC364	✓ ELEC365				
✓ ENGR213	✓ ELEC264	✓ ELEC364	✓ ELEC370				
✓ ENGR213	✓ ELEC264	✓ ENGR202	✓ ELEC370				
✓ ENGR233	✓ ENCS282	✓ ENGR202	✓ ENGR301				
✓ ENGR233							

Legend
✓ Completed
⊘ Prereq Needed
● Prereq Met
▸ In Progress

Figure 23.5 - Program Requirement screenshot

When the Program Requirement tab is clicked, the user gains access to the course sequence for their registered program as seen in Figure 23.5.

23.2.7 Sequence Planner

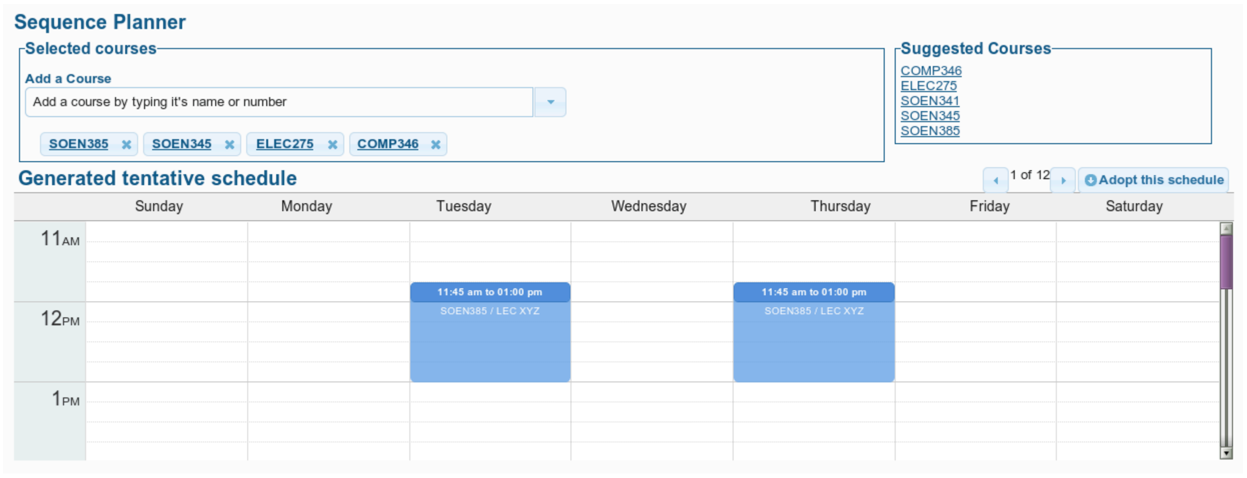


Figure 23.6- Sequence Planner Screenshot

When the Sequence Planner tab is clicked, the user gains access to the schedule generator as seen in Figure 23.6.

23.2.7.1 Adding /Removing a course

Upon accessing the Sequence Planner tab, the user is provided with a list of suggested courses to register for. These courses are taken from the user's program's specific course sequence. To add a course:

1) Type the name or code of the course.
2) Select the course you wish to view.
3) Click the 'Add' button.

To remove a course, click on the 'Delete' button that is shaped in the form of an X.

23.2.7.2 Cycling through Schedules

PlanZilla may provide user with more than one suggested schedule. To cycle through this schedule the left and right arrows on the top right of the schedule viewer.

23.2.7.3 Adding/Removing constraints

The user can add constraints such to the schedule so that the generator will know not to add a course during this time.

1) Select a timeslot on the calendar by clicking at the start time and dragging until the end time.
2) A window should appear. Type in the name of the constraint.

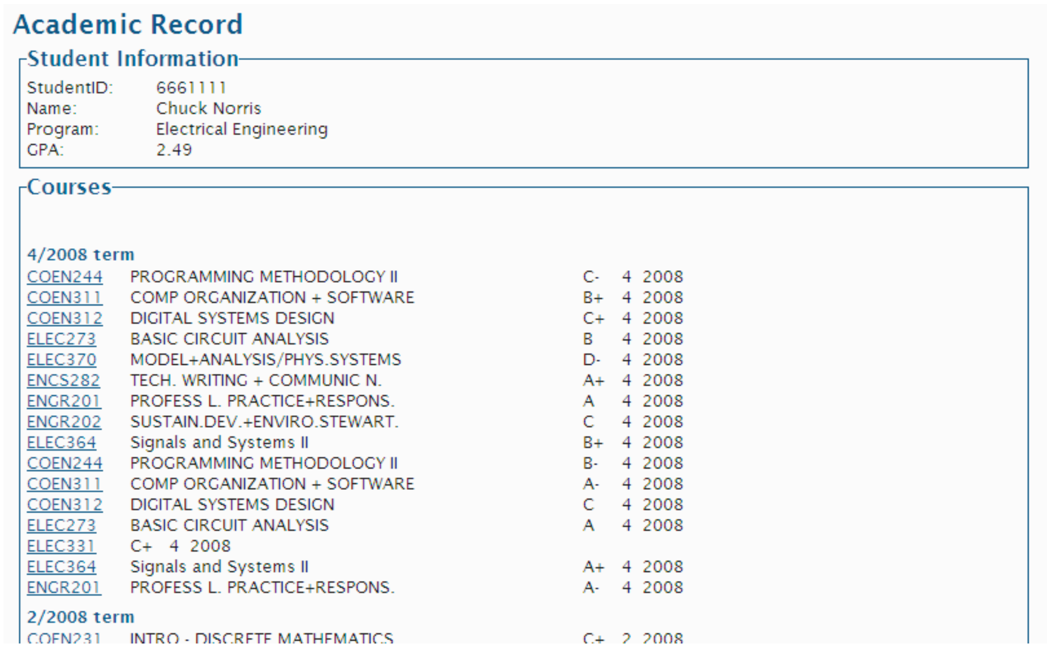
3) Either click on the ‘Ok’ button or press enter.

To remove a constraint, click on the ‘Delete button in the shape of an X.

23.2.7.4 Generate Schedule

Once all of the courses have been added, click on the ‘Generate Schedule’ button. The user can then use the left and right arrow buttons to view the different schedule options when more than one are available.

23.2.8 Academic Record



Academic Record

Student Information

StudentID:	6661111
Name:	Chuck Norris
Program:	Electrical Engineering
GPA:	2.49

Courses

4/2008 term

COEN244	PROGRAMMING METHODOLOGY II	C-	4	2008
COEN311	COMP ORGANIZATION + SOFTWARE	B+	4	2008
COEN312	DIGITAL SYSTEMS DESIGN	C+	4	2008
ELEC273	BASIC CIRCUIT ANALYSIS	B	4	2008
ELEC370	MODEL+ANALYSIS/PHYS.SYSTEMS	D-	4	2008
ENCS282	TECH. WRITING + COMMUNIC N.	A+	4	2008
ENGR201	PROFESS L. PRACTICE+RESPONS.	A	4	2008
ENGR202	SUSTAIN.DEV.+ENVIRO.STEWART.	C	4	2008
ELEC364	Signals and Systems II	B+	4	2008
COEN244	PROGRAMMING METHODOLOGY II	B-	4	2008
COEN311	COMP ORGANIZATION + SOFTWARE	A-	4	2008
COEN312	DIGITAL SYSTEMS DESIGN	C	4	2008
ELEC273	BASIC CIRCUIT ANALYSIS	A	4	2008
ELEC331	C+ 4 2008			
ELEC364	Signals and Systems II	A+	4	2008
ENGR201	PROFESS L. PRACTICE+RESPONS.	A-	4	2008

2/2008 term

COEN231	INTRO - DISCRETE MATHEMATICS	C+	2	2008

Figure 23.7- Academic Record Screenshot

When the Academic Record tab is clicked, the user gains access to all of their academic information. This includes their program, their GPA, their past courses – along with their corresponding grade – and their currently registered courses. A sample can be seen in Figure 23.7

REFERENCES

[1] "Browser Statistics." [Online]. Available: http://www.w3schools.com/browsers /browsers_stats.asp. [Accessed: Feb. 4, 2010].

[2] "jQuery: The Write Less, Do More, JavaScript Library." [Online]. Available: http://www.jquery.com/. [Accessed: Feb. 4, 2010].

[3] "JQuery Usage Statistics". [Online]. Available: http://trends.builtwith.com/javascript/ JQuery. [Accessed: Feb. 4, 2010].

[4] "JavaScript Usage Statistics". [Online]. Available: http://trends.builtwith.com/javascript. [Accessed: Feb. 4, 2010].

[5] "Developer's Guide – Google AJAX Libraries API". [Online]. Available: http://code.google.com/intl/xx-pirate/apis/ajaxlibs/documentation/. [Accessed: Feb. 4, 2010].

[6] "Home – jquery-week-calendar – GitHub." [Online], Available: http://wiki.github.com/robmonie/jquery-week-calendar/. [Accessed: Mar. 5, 2010].

APPENDIX A – Latest Requirement Test Report

PLANZILLA REQUIREMENT & QA CHECKLIST

Checklist version 1.0.5

Build Version:	1.4111	Tested by:	Phuong-Anh-Vu Lai
		Date :	070410
Test platform:	Windows 7 - Google Android		
Additional Info:			

Test Status:	FAIL - Build fails for multiple requirement issues.
Build Status:	To be revised by developers.
Comments:	Pre-release alpha testing build.

Section 1 : Functional Requirements

These test cases are related to actual functionalities within the application. They can be tested while using the application normally. Each case is related to a corresponding user case. For more detail about the test consult the related user case from the Scope, Requirement and Plan document. — **Pass**

Case ID	Test Description	Related UC	Result
1.1	**Initial Start Up**		**Pass**
1.1.1	User is able to launch the web application using a web browser entering this address : (https://team7.ath.cx/)	UC1	**Pass**
~~1.1.2~~	~~If a non-optimal browser was used to access this application, a warning message will appear but will let user continue.~~	~~UC1~~	**N/A**
~~1.1.3~~	~~If a non-supported browser was used to access this application, this application will display an error and will not launch.~~	~~UC1~~	**N/A**
1.1.4	The application launches succesfully and displays the correct initial screen : View schedule.	UC1	**Pass**
1.2	**Authentication**		**Pass**
1.2.1	User is able to access a login page or module.	UC2	**Pass**
1.2.2	User can enter credentials for login.	UC2	**Pass**

1.2.3	If credential is valid, the user will be succesfully logged in.	UC2	**Pass**
1.2.4	If an invalid credential was used, the system will display a error message and ask user to try again.	UC2	**Pass**
1.2.5	While being logged in, the user can logout from the application.	UC3	**Pass**
1.2.6	Accessing "Log out" sucessfully logs the user out of his account.	UC3	**Pass**
1.2.7	After having succesfully logged out, the screen no longer display any information related to user's session or account and is taken back to default page.	UC3	**Pass**
1.3	**Schedule Viewer**		**Pass**
1.3.1	View Schedule button is available on the navigation bar.	UC4	**Pass**
1.3.2	View Schedule page is the default page after logged in.	UC4	**Pass**
1.3.3	A schedule is displayed on screen.	UC4	**Pass**
1.3.4	The schedule viewer correctly displays the current saved schedule in user's account. If no schedule has been set, that should also be reflected correctly on the viewer.	UC4	**Pass**
1.4	**Room / Professor Schedule**		**Pass**
1.4.1	From the Schedule Viewer, user can access specific room and professor schedules.	UC5	**N/A**
1.4.2	The Schedule viewer displays the schedule correctly.	UC5	**N/A**
1.5	**Course Browsing**		**Pass**
1.5.1	Browse Course List is available on the navigation bar.	UC6	**Pass**
1.5.2	Clicking on Browse Course List button will display the correct page.	UC6	**Pass**
1.5.3	From this page, user can select a departement and course number from a list.	UC6	**Pass**
1.5.4	Upon selecting a course, the correct information for that course is displayed. Browsing through courses does not yield any errors.	UC6	**Pass**

1.6	**Program Sequence**		**Pass**
1.6.1	Program Sequence button is available from the navigation bar.	UC7	**Pass**
1.6.2	Clicking on Program Sequence will display the correct page.	UC7	**Pass**
1.6.3	A complete list of user's program course sequence is displayed correctly.	UC7	**Pass**
1.6.4	If the user is not enrolled in a program, an warning should be displayed to inform user that a sequence is not available.	UC7	**Pass**
1.7	**Academic Record**		**Pass**
1.7.1	Academic Record is available from the navigation bar.	UC8	**Pass**
1.7.2	Clicking on Academic Record will display the correct page	UC8	**Pass**
1.7.3	User's academic record page correctly displays previously and currently taken course and displays their grade correctly.	UC8	**Pass**
~~1.8~~	**~~Activity Log~~ (Scoped out)**	-	**~~N/A~~**
~~1.8.1~~	~~Activity Log is available from the navigation bar.~~	~~UC8~~	**~~N/A~~**
~~1.8.2~~	~~Clicking on Activity Log will display the correct page~~	~~UC8~~	**~~N/A~~**
~~1.8.3~~	~~User's academic record correctly displays logged transactions from user's activity in the system.~~	~~UC8~~	**~~N/A~~**
1.9	**Schedule Planner**		**Pass**
1.9.1	A Schedule planner is available from the navigation bar.	UC9	**Pass**
1.9.2	Clicking on Schedule planner will display the correct page.	UC9	**Pass**
1.9.3	The page contain a Generate Schedule button.	UC9	**Pass**
1.9.4	Pressing this button will offer user a list of possible schedules.	UC9	**Pass**
1.9.5	User is able to cycle through different schedule and choose a specific one.	UC9	**Pass**
~~1.9.6~~	~~The application will prompt for confirmation when a schedule is selected.~~	~~UC9~~	**~~N/A~~**

~~1.9.7~~	~~Once user has confirmed, the system confirms that the operation was successful.~~	~~UC9~~	**~~N/A~~**
1.9.8	Schedules are displayed correctly of the viewer.	UC11	**Pass**
1.9.9	While user is selecting schedule, the application offers the option to add time constraint or preferences.	UC11	**Pass**
1.9.10	Any changes in the constraints and preferences are reflected on the schedule viewer.	UC11	**Pass**
~~1.9.11~~	~~Once a schedule is successfully selected. All changes and scheduling are saved and will be displayed the next time user logs in.~~	~~UC11~~	**~~N/A~~**
1.9.12	An error is generated if application cannot find a suitable schedule for the user according to his constraints.	UC11	**Pass**

Section 2 : Non-Functional Requirements		
This section contains qualitative requirements and is not tied with a specific functionality. Should these tests fail at any point in the application, it automatically fails.		**Pass**
Case ID	Test Description	**Result**
2.1	The application responds quickly. Any regular requests should take less than 2 seconds except for schedule generation. (Req. Section 3.3.1)	**Pass**
2.2	Generating schedules should not take more than 5 seconds. (Req. Section 3.3.1)	**N/A**
2.3	The system is user-friendly and intuitive. User should never be lost while navigating through application and can easilly find sections. (Req. Section 3.3.2)	**Pass**
2.4	Each page contain a back button which takes user to the last visted page. (Req. Section 3.3.2)	**N/A**
2.5	The system should easily scalable. (Can only be tested by certified a developer. Pass by default if build is released for testing) (Req. Section 3.3.3)	**Pass**
2.6	System should support a minimum of 5000 concurrent users. (See Stress testing results) (Req. Section 3.3.3)	**Not tested**
2.7	If system cannot log user in due to large traffic, the application should return a error message to try again later. (Req. Section 3.3.3)	**Not tested**

2.8	System is secure against security threats (see Security Testing results). (Req. Section 3.3.4)	**Pass**
2.9	User with disabilities should be able to use the application with accessibility features without any problems. (Req. Section 3.3.5)	**Pass**
2.10	The application is printer friendly. Printing any page should not yield unwanted artefacts or extra white pages. (Req. Section 3.3.6)	**Pass**
2.11	User is able to resume a session without losing input data after a wireless connection is lost. (Req. Section 3.3.7)	**Pass**
2.12	The application is free from graphical issues.	**Pass**

APPENDIX B – Latest Unit Testing Reports

PLANZILLA UNIT TESTING CHECKLIST

Checklist version 1.0.0

COMPONENT :	Authentication

Compon. Version:	1.0.0	Tested by:	Andreas Eminidis & Phuong-Anh-Vu Lai
Build Version:	0.3.0	Date :	070410
Test platform:	Windows 7 – Chrome – Notepad - SkipFish		
Additional Info:	PASS - Authentication Module passes tests for build 0.3.0		

Section 1 : Design Requirement

This section contains test cases related to the presence of requirement and functionalities of the module as described in the design document.

Case ID	Test Description	Result
1.1	The Authentication module can communicate with the Main module and the MySQL database without any issues	Pass
1.2	The Authentication module forces clients to have proper access rights for each module by requesting them to login using their credentials.	Pass
1.3	ValidateCredentials is implemented.	Pass
1.4	ValidateUsername is implemented.	Pass
1.5	ValidatePassword is implemented.	Pass
1.6	CheckUserExistence is implemented.	Pass
1.7	Login is implemented.	Pass
1.8	Logout is implemented.	Pass
1.9	CreateUser is implemented.	Pass
1.10	DeleteUser is implemented.	Pass
1.11	ChangePassword is implemented.	Pass
1.12	$AuthenticationLevel is a implemented as a attribute.	Pass

Section 2 : Implementation Test		
This section contains test cases related to the actual functionality of the module. List all functions and related functionalities and test if they work as intended. These will test if they accept correct input and yield the corresponding right output. If applicable, error handling functionalities such as incorrect inputs should also be listed.		
Case ID	Test Description	Result
2.1	ValidateCredentials works as intended by design.	Pass
2.2	ValidateUsername works as intended by design.	Pass
2.3	ValidatePassword works as intended by design.	Pass
2.4	CheckUserExistence works as intended by design.	Pass
2.5	Login works as intended by design.	Pass
2.6	Logout works as intended by design.	Pass
2.7	CreateUser works as intended by design.	Pass
2.8	DeleteUser works as intended by design.	Pass
2.9	ChangePassword works as intended by design.	Pass
2.10	$AuthenticationLevel works as intended by design.	Pass

Section 3,4 and 5 : Security Test		
These tests cases are derived from Google's Security Testing Web application Skipfish.		
Case ID	Test Description	Result
3	**High risk flaws (potentially leading to system compromise)**	**Pass**
3.1	Server-side SQL injection (including blind vectors, numerical parameters)	Pass
3.2	Explicit SQL-like syntax in GET or POST parameters.	Pass
3.3	Server-side shell command injection (including blind vectors).	Pass
3.4	Server-side XML / XPath injection (including blind vectors).	Pass
3.5	Format string vulnerabilities.	Pass
3.6	Integer overflow vulnerabilities.	Pass
3.7	Locations accepting HTTP PUT	Pass
4	**Medium risk flaws (potentially leading to data compromise)**	**Pass**
4.1	Stored and reflected XSS vectors in document body (minimal JS XSS support present).	Pass
4.2	Stored and reflected XSS vectors via HTTP redirects.	Pass

4.3	Stored and reflected XSS vectors via HTTP header splitting.	Pass
4.4	Directory traversal (including constrained vectors).	Pass
4.5	Assorted file POIs (server-side sources, configs, etc).	Pass
4.6	Attacker-supplied script and CSS inclusion vectors (stored and reflected).	Pass
4.7	External untrusted script and CSS inclusion vectors.	Pass
4.8	Mixed content problems on script and CSS resources (optional).	Pass
4.9	Incorrect or missing MIME types on renderables.	Pass
4.10	Generic MIME types on renderables.	Pass
4.11	Incorrect or missing charsets on renderables.	Pass
4.12	Conflicting MIME / charset info on renderables	Pass
4.13	Bad caching directives on cookie setting responses.	Pass
5	**Low risk issues (limited impact or low specificity)**	**Pass**
5.1	Directory listing bypass vectors.	Pass
5.2	Redirection to attacker-supplied URLs (stored and reflected).	Pass
5.3	Attacker-supplied embedded content (stored and reflected).	Pass
5.4	External untrusted embedded content.	Pass
5.5	Mixed content on non-scriptable subresources (optional).	Pass
5.6	HTTP credentials in URLs.	Pass
5.7	HTML forms with no XSRF protection.	Pass
5.8	Bad caching directives on less sensitive content.	Pass

PLANZILLA UNIT TESTING CHECKLIST

Checklist version
1.0.0

COMPONENT :	Database

Compon. Version:	1.5.0	Tested by:	Marc-André Moreau & Phuong-Anh-Vu Lai
Build Version:	1.411	Date :	100410
Test platform:	Multiple platforms - Windows 7/Linux - Firefox/Chrome		
Additional Info:	PASS – Database module is functioning as intended.		

Section 1 : Design Requirement		
This section contains test cases related to the presence of requirement and functionalities of the module as described in the design document. (List all of them or major groups if too many)		
Case ID	Test Description	Result
1.1	Database handles storing course information such as the department, course number, credits, course description, different classes for the same course with a time and a teacher.	Pass
1.2	Database is able to store student and teacher information.	Pass
1.3	Database is able to represent teachers teaching different classes and students being registered in different classes.	Pass

Section 2 : Implementation Test		
This section contains test cases related to the actual functionality of the module. List all functions and related functionalities and test if they work as intended. These will test if they accept correct input and yield the corresponding right output. If applicable, error handling functionalities such as incorrect inputs should also be listed.		
Case ID	Test Description	Result
2.1	Obtaining course information	Pass
2.2	Obtaining student information	Pass

Section 3 : Other Test		
Feel free to add more section if necessary to have a well rounded test checklist that checks your module.		
Case ID	Test Description	Result
3.1	SQLInjectMe does not detect any SQL injection points	Pass
3.2	Nikto does not find any relevant security issues in the web application	Pass