
# ASHESI UNIVERSITY COLLEGE

## GENERAL COURSE SCHEDULING APPLICATION

**APPLIED PROJECT**

B.Sc. Computer Science

**Ernest Kufuor Jr.**

**2017**

**ASHESI UNIVERSITY COLLEGE**

# General Course Scheduling Application

**APPLIED PROJECT**

Applied Project submitted to the Department of Computer Science, Ashesi University College in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Science

**Ernest Kufuor Jr.**

**April 2017**

# Declaration

I hereby declare that this Applied Project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

……

Candidate's Name:

……

Date:

……

I hereby declare that preparation and presentation of this Applied Project were supervised in accordance with the guidelines on supervision of Applied Project laid down by Ashesi University College.

Supervisor's Signature:

……

Supervisor's Name:

……

Date:

……

## Acknowledgment

I want to thank the God Almighty for helping me complete this applied project. I would like to thank my mom and dad, who did their best to bring me up to this point in life and give me that privilege of having quality education. I would also like to give a lot of credit to Mr. David Sampah, Jane Amerley Annan and Stephanie Belnye for giving me hope and support when I needed the most. Finally, I would like to thank all the people who supported me in completing this Applied Project.

# Abstract

Heuristic algorithms have been known to be the most common and effective methods to solving the classical course scheduling problem which has been found to be a non-deterministic polynomial time hard problem. Many software systems have been built on these algorithms to provide a standard and working solution. This paper seeks to look at the gaps of these solutions and propose a newer one which would potentially simplify the process of course scheduling and provide an even more effective solution within the scope of Ghana. This paper provides a course scheduling web application which implements a genetic algorithm as a solution.

# Table of Contents

# Chapter 1: Introduction

## 1.1 Background Information

Course scheduling is an essential part of a university's core activities to ensure that students get the best out of their lectures and lecturers can work under ideal and efficient conditions. Course Scheduling is basically the act of organizing lectures on a specified timetable that satisfies the recognized constraints (Wasfy & Aloul, 2006). The main issues with course scheduling arise when certain constraints are not satisfied. The constraints may include:

- A lecturer not being assigned to more than one lecture hall at the same time.
- A student not having more than one course at the same time.
- A lecture hall not hosting more than one course at the same time.
- More recently, a lecturer not having three or four courses assigned to him or her running concurrently.

In some cases, some schools prefer to use online software's to automate a course schedule. This can sometimes slow down the process of drawing up a schedule but usually does not provide an optimal solution or a working solution since certain constraints may vary from institution to institution. Most often, schools prefer to manually draw up the schedule using simple desktop tools like Microsoft Office or traditional methods like writing out the schedule on paper. This often provides a suitable schedule and conforms with the constraints of the set institution but is a very cumbersome and tiresome process. More recently, the issue of lecturers been overloaded with courses has risen and most of these techniques do not cater for this (Dahiya, 2015). Regardless of the number of courses a student takes, it can be assumed that not all the courses would be offered on the same day.

This only means that a student would have enough time to draw up a personal schedule for himself or herself and hence course overloading on the part of the student is not seen as a constraint.

In 2000, the school-age population in Africa grew to 334 million people. (United Nations, 2003) This increase in population has led to the enlargement of educational institutions, Personnel, infrastructure, and courses have also increased in number or size as a response to the population increase. For instance, Ashesi University College has discussion classes which differ in time length from normal lecture classes. Hence a schedule for Ashesi University College would differ from another school that uses a different system. As such, an automated system with preference to lectures scheduling is key to school's today.

## 1.2 Project Overview

Growing colleges usually face the problem of course scheduling due to many inconsistencies. These inconsistencies are usually with the number of lecturers available in a semester, the growing number of students of the school each year and the number of lecturer halls available in a semester. Usually, because of the inconsistency of the number of lecturers, there might be the case where a lecturer may be required to teach more courses than he usually does in a semester. This situation usually leads to a lecturer having back to back classes which can be heavy on the lecturer.

This project seeks to build upon research done by a previous Ashesi University College student, which used a particle swarm optimization algorithm to provide an automated course scheduling system. This project hopes to extend the implemented algorithm and build a graphical user interface for the algorithm thereby building a fully

functional system for general use. This project would be a web application making use of Hypertext Preprocessor (PHP), JavaScript, and Hypertext Mark-up Language (HTML).

## 1.3 Objective

The objective of the project is to build an application that implements a course scheduling algorithm. The application should be able to generate an optimal course schedule based on given input. The schedule should hopefully be, conflict free and should be usable by elementary schools, high schools, or universities.

## 1.4 Importance of the System and Motivation

Automated Systems have been increasingly popular and useful over the ages. Going back to the years of industrial revolution, it was evident that work was made easier with a reduced dependency on manpower and an increase in the precision of outcomes. (Mingo, 2000) Course scheduling, with all its hassle and complexity, is a problem that can be and should be solved with an automated system. Course Scheduling has been found to be a non-deterministic polynomial-time hard (NP-hard) problem and hence has no fixed or definite solution. (Hamalainen, 2006) Many applications implementing various algorithms have been developed to tackle course scheduling though they do not tackle the issue of lecturer overloading.

# Chapter 2: Related Work

## 2.1 Overview

This section provides information on works related to course scheduling and identifies the limitations of the works and conceivably states how the work contributes towards the solution to the problem identified.

## 2.2 Related Works

The_classical_course scheduling problem is considered as a non-deterministic polynomial time hard problem (Hamalainen, 2006) It has no definite solution and can have several solutions. Not all solutions may be optimal. The nature of this problem highlights the difficulty to attain an optimal solution given that each solution does not guarantee optimization. Generally, heuristic algorithms like *genetic algorithms* and *tabu search* have been used to tackle the solution though each algorithm comes with its own benefits like computation speed or number of allowable variables.

Karami and Hasanzadeh proposed a new hybrid genetic algorithm to the non-deterministic polynomial time (NP Complete) problem of course scheduling (Karami, & Hasanzadeh, 2012). Their algorithm used population-based metaheuristics algorithms and evolutionary algorithms. The initial population is stored into a red-black tree data structure, to which after their Hybrid genetic algorithm creates children from previous individuals by its operators. The algorithm used Hill climbing to improve their results. The results of their algorithm were impressive with regards to the algorithms convergence rate which proposed a solution after 200 iterations. Karami and Hasanzadeh's hybrid genetic algorithm was time efficient but could not find a feasible solution for large instances. This seems to be a major drawback because even though time efficiency is key when it comes to course scheduling,

it is important to find a feasible solution to satisfy all constraints. It is necessary that an optimal algorithm should be able to satisfy larger university schedules.

Castrillón's article considers how evolutionary algorithms and cognitive rhythms can be used to tackle the university's course scheduling problem and draw up an optimal timetable (Castrillón, 2015). His algorithm suggests a new and improved algorithm based on an evolutionary algorithm and students' cognitive rhythm. After the tests and comparisons, it was found that the algorithm proposed was 15% more efficient than the already existing algorithms and shows more system stability. In choosing the right software to look at to help solve the course scheduling problem, efficiency is key. Even though Castrillón's proposed algorithm tends to be more efficient, it does not fully satisfy the context of Ashesi's problem. The solution presented takes into consideration students' cognitive rhythms but not the work load on the lecturer. A student in Ashesi's context, regardless of the schedule would still have time to indulge in his or her personal activities. The problem arises when lecturers have consecutive classes on some days and end the day exhausted.

A more ideal approach was found in Dahiya Saddharth's solution to course scheduling. It considers scheduling while taking into consideration the preference for course and professors (Dahiya, 2015). His solution presents a hybrid genetic algorithm called course scheduling with preference optimization. The algorithm considers the various preferences and constraints and return a valid schedule where both the preferences for both professors and courses had been maximized. The main limitation stated by Saddharth in his thesis was the decrease in the algorithm's run-time when there was an increase in courses and faculty. This may seem a big drawback when it comes to large universities but for small universities, it would be an optimal solution.

Over the years, several applications have been developed to tackle the problem of course scheduling making use of different algorithms. From articles, the most common algorithms used are genetic algorithms, swarm intelligence algorithms and hybrid algorithms. These algorithms offer better computation time and seem to offer more optimal solutions than other algorithms when it comes to course scheduling. Applications on the market, be it open source or not, usually implement these algorithms.

Event Management System (EMS) software has been found to be a popular scheduling software that offers optimal and institution tailored schedules. It offers a premium service, where users are more of clients to the software and would have to work with the administrators of the EMS software. Even though, effective, this process may be inconvenient for users who would prefer a free and do-it-yourself process. The College Scheduler by Civitas Learning bears the same similarities with the EMS software.

Open source and free applications like Unitime, ScheduleIt and CyberMatrix Course Scheduler offer decent course scheduling functionality. The main problem with these is the fact that schedules usually clash when slightly larger courses and lecturers are considered. Because they are general purpose, some of the applications are quite complex to understand and to further customize with additional functionalities.

Ashesi University College has seen instances of solutions to its course scheduling problem though none of them have proved to stand as a standard solution. The most recent of which was proposed in 2015. The solution was an open-source application called Free Timetabling Software (FET) (Papafio, 2015). The Registry complained of the application not being able to satisfy all its constraints and hence could not provide an optimal schedule.

Another project only provided a theoretical solution using a particle swarm optimization algorithm for Ashesi's course scheduling problem. The algorithm took into

consideration lecturers preferences and was yet to be extended into a full application for testing with real life variables. Mathematically, this approach was shown to be more effective than other traditional algorithms like constraint programming, tabu search and simulated annealing. (Agbenowosi, 2014)

Similar to Agbenowosi's, George Donkor looked at providing an effective algorithm to solve Ashesi University's course scheduling problem. He compared different algorithms based on their computational time, ease of implementation, solution quality and constraint handling and concluded that the Particle Swarm Optimization was the best in the set criteria. He concluded that the Particle Swarm Optimization was an easy algorithm to implement which produced good results. (Donkor, 2014)

From the above, several solutions with different algorithms have been trialled and tested. The problems with these applications differ by institution and user, either it is not convenient or is not able to satisfy the school's constraints. These solutions also tend to be very rigid and cumbersome when editing and entering data.

To tackle this problem, a software which properly combines automation and manual usage would be an ideal solution. In simplifying the process and improving upon the user interface, an ideal and usable solution can be developed with a simple genetic algorithm working in the backend. By context, this solution may also serve as a general-purpose application to many schools in Ghana as existing online applications may either be too complex or cannot provide an optimal schedule.

# Chapter 3: Requirements Specification

## 3.1 Overview

This section provides an overview of the requirement chapter and a scope of this project.

## 3.2 Project Scope

In recent times, optimization algorithms, like particle swarm optimization and genetic, have been found to be popular approaches to solve non-deterministic polynomial-time complete problems as they provide an optimal solution depending on the given problem. The chosen solution makes use of an open-source genetic algorithm tailored towards reducing schedule overloading on lecturers. This project seeks to interpret and build a graphical user interface for this algorithm. With respect to this, an understanding of the constraints, preferences and requirements are necessary. Throughout this section, the application would be referred to as the Beehive Scheduling Application.

## 3.3 Overall Description

The Beehive Scheduling Application is meant to be a generalized application software that can provide course scheduling features with preference to teaching times. It is meant to be a web application.

## 3.4 Requirements Gathering

The purpose of this project is to provide a graphical user interface for a genetic algorithm to resolve the course scheduling problem in schools in Ghana using Ashesi University as its context. To get a suitable algorithm, the application must satisfy the school's preferences, constraints, and requirements. As such, conversations would be held with the registrar's office to draw knowledge from the existing solution to outline the user and system requirements needed for course scheduling in general.

## 3.5 Use Case

This sections looks at a scenario and a use case diagram which highlight the user's interaction with the system and help define the user's requirements.

**Scenario**: Herty is a school registrar who spends far too much time when scheduling courses for the semester. She wishes to be able to automate the process with an easy to use application. She wants to be able to create a schedule based on input categorized by different lecturers, rooms, courses, and time slots. When the schedule is generated, she wants to be able to make changes to her schedule and view the schedule in an event calendar.

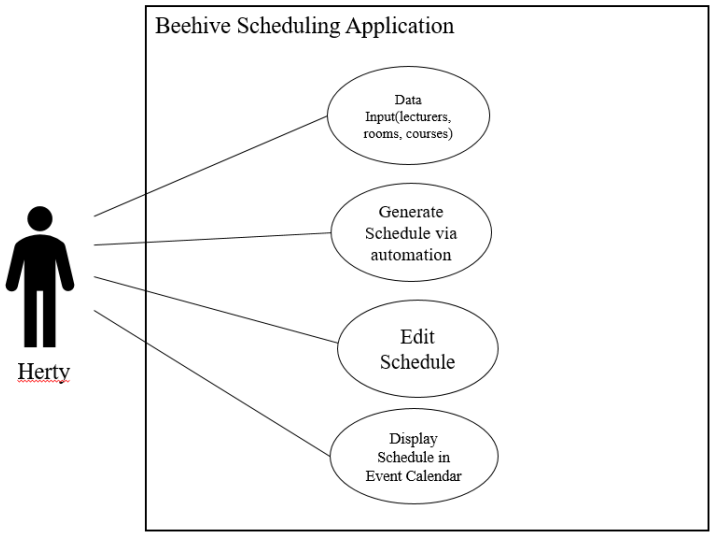


Figure 3.1: Use Case Diagram

## 3.6 Product Features / User Requirements

The Beehive Scheduling Application seeks to provide generalized course scheduling functionalities for general use via an effective algorithm. The Beehive Scheduling Application should provide services to actors who use it. Therefore, the application should be able to take in input and give output. The *actor* or *user* of this system is any user who is

found using the system for course scheduling purposes, be it a school registry or an individual.

The input should be in the form of available lecturers, available rooms, lecture times, available courses and course hours. The user should be able to generate a schedule based on the parameters supplied, and view the resultant schedule. The Beehive Scheduling Application should also return necessary reports and feedback to the user, in the event of an error. The Beehive Scheduling Application should also be able to take in input as a file, read the contents of file and distinguish between the attributes or entities within the file that are necessary to produce a time table. The application should be able to give an error report where necessary.

## 3.7 Functional Requirements

This section provides the stripped-down functional requirements of the scheduling software. All of these functionalities would be addressed in this application. The major functional requirements of the application can be found below:

a) Data Input by user / Adding data to the database.

b) Schedule Creation.

c) Schedule Display.

### 3.7.1 Data Input by User

**Description***:* The user inputs the data necessary to implement a schedule (rooms, courses, and lecturers).

**Inputs**: *Course*: name, number, department; *Room*: number, department; *Lecturer*: name, department, respective courses;

**Output**: Confirmation or error message.

**Classes to Implement:** User class (main class), adb class

**Methods to Implement**: addCourse, addRoom, addLec, getRoom, getLec, getCourse, addRoomGroup, addCourseGroup, addLecGroup, delLec, delRoom, delCourse

**Test Plan:** The test plan would be done using unit testing via a single user.

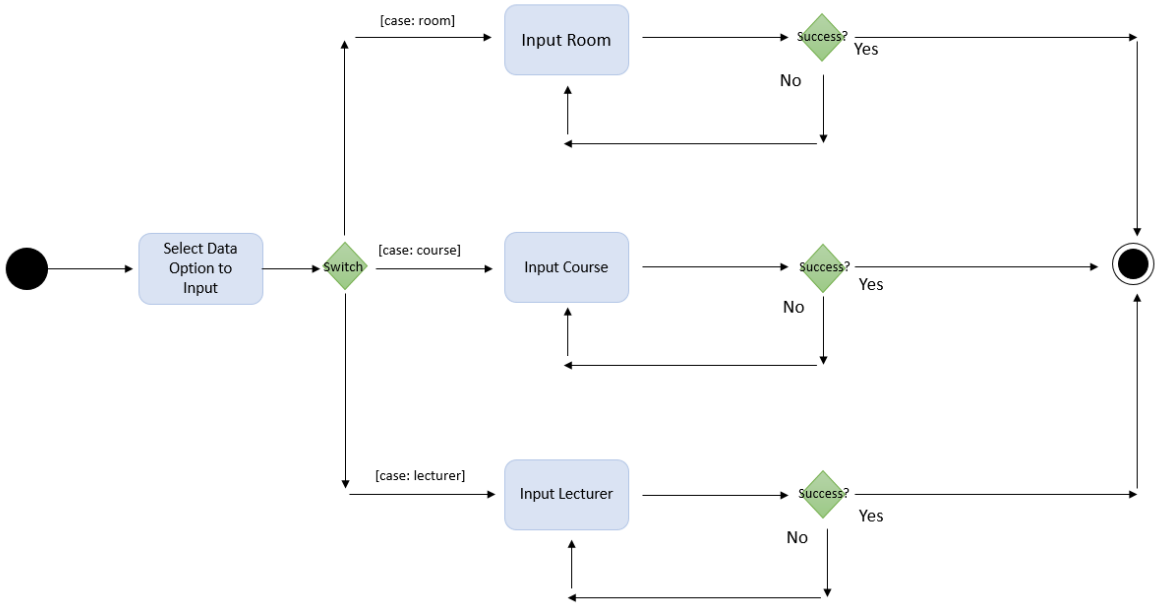


Figure3.2: Activity diagram for data input

### 3.7.2 Schedule Creation

**Description***:* Application creates a schedule using data in the database, necessary parameters and a genetic algorithm.

**Inputs**: Course name; room number, lecturer name, days, timeslots, year group, term, day start, day end, room type.

**Output**: Confirmation or error message.

**Classes to Implement:** schedule, room, course, lecturer, department, algorithm, population, data, entry and meetingTime

**Test Plan:** The test plan would be done using unit testing via a single user. Sample schedules would be created to see whether an optimal schedule can be created.

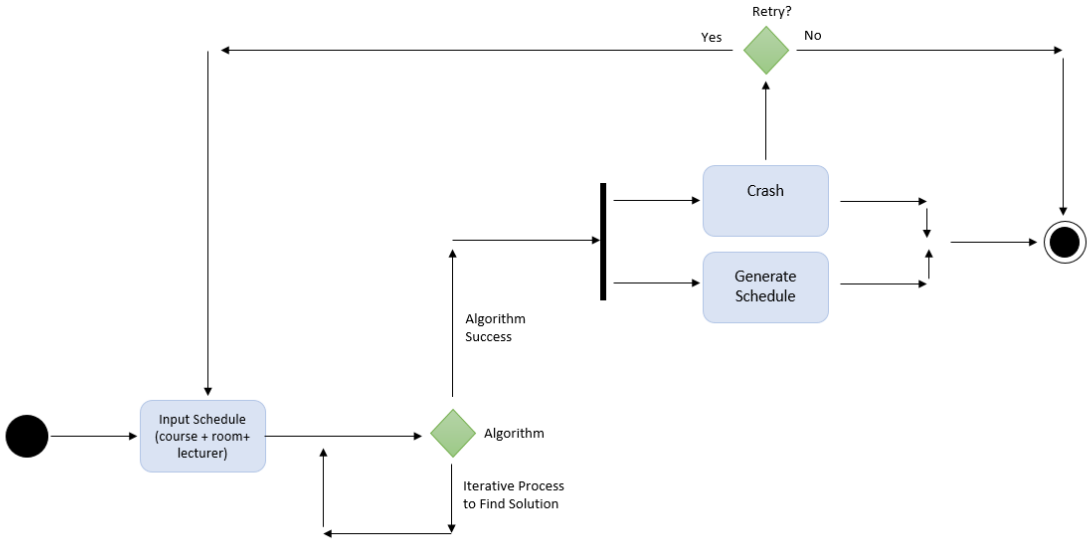


Figure 3.3: Activity diagram for schedule creation

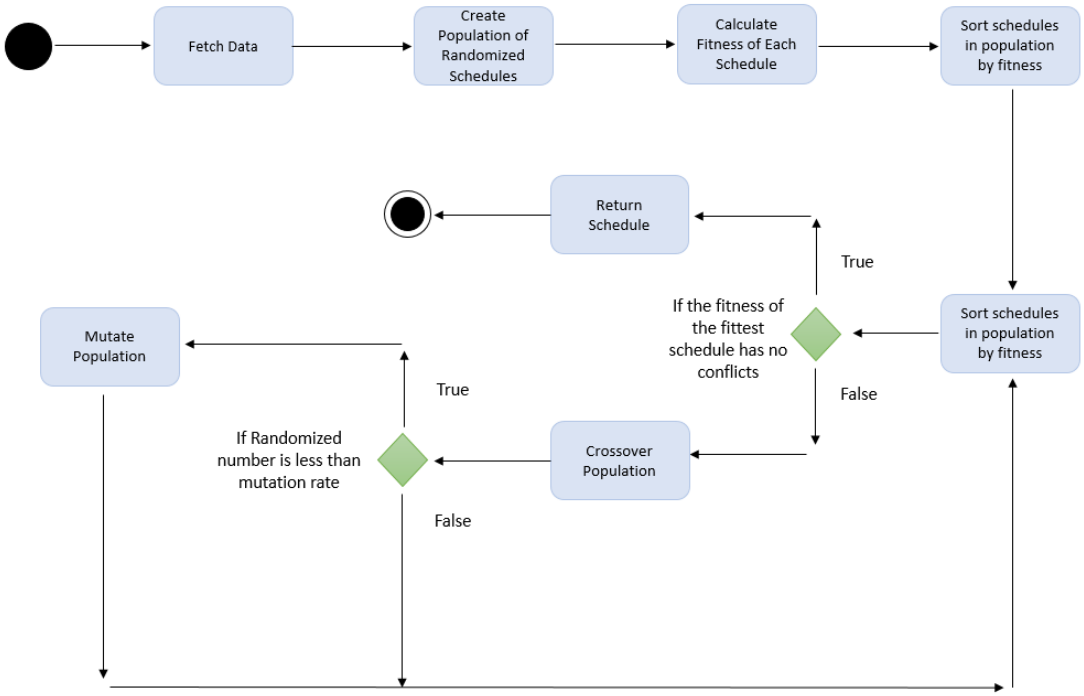


Figure 3.4: Activity diagram for algorithm

### 3.7.3 Schedule Display

**Description***:* Application fetches data from the database and displays it on an event calendar.

**Inputs**: none.

**Output**: Event Calendar filled with schedules or empty event calendar.

**Classes to Implement:** user class

**Methods to Implement**: getSchedules

**Test Plan:** The test plan would be done using unit testing via a single user. Sample schedules would be created to see whether an optimal schedule can be created.

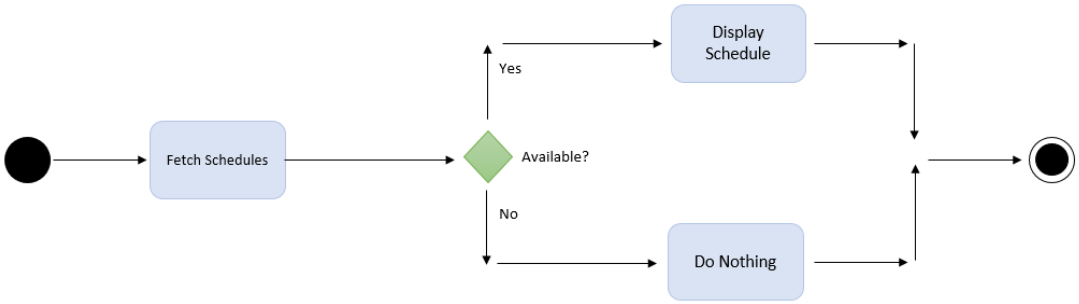


Figure 3.5: Activity diagram for schedule display

## 3.8 Additional Features

This section looks at additional features that the system would have.

- The Beehive Scheduling Application should be able to return error scripts and logs in a human readable format.
- The user should be able to edit information within the system where necessary.

## 3.9 Constraints

The Beehive Scheduling Application would have to observe certain constraints to fully produce an optimized schedule for a user.

- A lecturer cannot have two lectures at the same time.
- More than one lecture cannot occur at the same time in the same room.
- Two courses involving the same group of students cannot be taught at the same time.
- A lecturer should not have an overloaded schedule.

## 3.10 Stakeholders

This section highlights the various entities who would be affected by the usage of the application.

1. **Lecturers:** Basically, the course scheduling provides a schedule for which lecturers can follow in their work routines.
2. **Students:** Student's would follow the schedule to know their course times and room numbers.
3. **Academic Registry / Department of a school:** For a university or a school, this department would use the Beehive Scheduling Application to create the schedule that the students and lecturers would follow.
4. **Organizations / Institutions:** An institution with vested interest in drawing up a calendar schedule for its workers can make use of the Beehive Scheduling Application or any other department in the school that organizes sessions based on the school's course schedule times.

## 3.11 Dependencies

This Beehive Scheduling Application would be built on and depend on:

- Computer systems
- User Input

## 3.12 Non-Functional Requirements

The non-functional requirements for the Beehive Scheduling Application are as follows:

- ***Performance / Speed***

  The software would need to produce an optimized schedule or response within an appropriate time.

- ***Security and Privacy***

  The software would need to provide some form of security and privacy for the users, to protect the information or schedule they have provided.

# Chapter 4: Architecture and Design

This chapter outlines the architecture and design of the Beehive Scheduling Application. It takes into perspective the different layers of the application and the various components, languages and tools needed to develop the application.

## 4.1 Overview

The graphical interface consists of a web based software that implements a genetic algorithm that finds an optimal schedule. The software is a web application and will be built on PHP: Hypertext Preprocessor (PHP), Cascading Style Sheets (CSS), JavaScript, and Hyper Text Markup Language (HTML). A Structured Query Language (SQL) database will be used to store the applications data.

## 4.2 Architecture

The project makes use of these languages because of researcher proficiency in these languages and the large community of developers that support the language. That being said, development and debugging is made easier.

The Beehive Scheduling application is implemented using a three-tier architecture, a type of client-server architecture. This architecture separates the application (business logic and data management), data (data storage), and presentation logic (graphical user interface) of the system into three separate components or layers. This architecture properly defines the functions and operations of each layer and hence debugging and testing of each layer is easier.

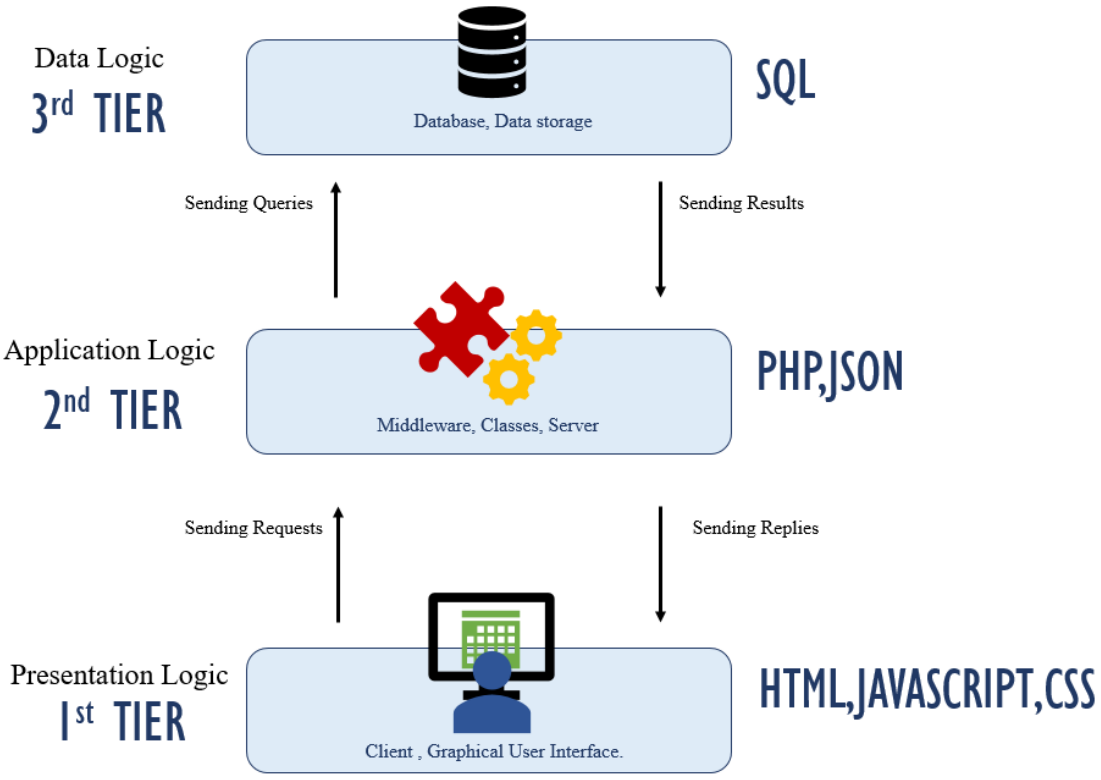


Figure 4.1: High-Level Architecture (Three Tier)

### 4.2.1 Presentation Logic / Layer

This is the topmost tier in the three-tier architecture. It consists of the user interface that enables the user to communicate with the application by translating the results and tasks of the system into a form the user can understand. (Sommerville, 2011) The frontend of the system is divided into two parts: an interaction screen and a client side which sends requests to the application layer. This layer guides the user through data input, schedule generation and display.

The Beehive Scheduling Application's user interactive frontend is built in HTML and CSS. The client side that sends requests to the application layer is built in JavaScript. The software makes use of frameworks to enhance performance, look and experience. These

templates include; Bootstrap, Creative Tim's Paper Dashboard and Intensify by TEMPLATED, all under the Creative Commons Public License. The event calendar is a JavaScript library plugin created by DayPilot. The client side communicates with the application layer by sending Asynchronous JavaScript and XML (AJAX) requests.

Data sent from the data layer is finally visualized here. The user's desire to visualize the schedule is done in this layer. The user interacts with the application at this level and can put in necessary data needed by the application to be able to generate a schedule.

### 4.2.2 Application Logic / Layer

The application layer serves as the middleware between the presentation layer and data layer. It holds the main logic of the application defined in classes and files. (Sommerville, 2011) It also contains the logic needed to send queries to the data layer and retrieve data. It returns necessary data, if requested, to the presentation layer to display to the user.

The algorithm of the application is found in this layer, which is the most important aspect of the application. This is where the automatic assigning of courses into slots would take place using the genetic algorithm. The algorithm is an open source Java code translated into JavaScript. This layer contains classes with sets of functions necessary for the operation of the application and for data management. Requests from the user to add, insert, delete, update, or fetch data is channelled from the presentation layer to the data layer through the application layer. The main language used by the application layer is PHP. It replies to requests made by the client via Java String Object Notation (JSON). Data management is achieved via SQL Queries sent from the application layer to the data layer.

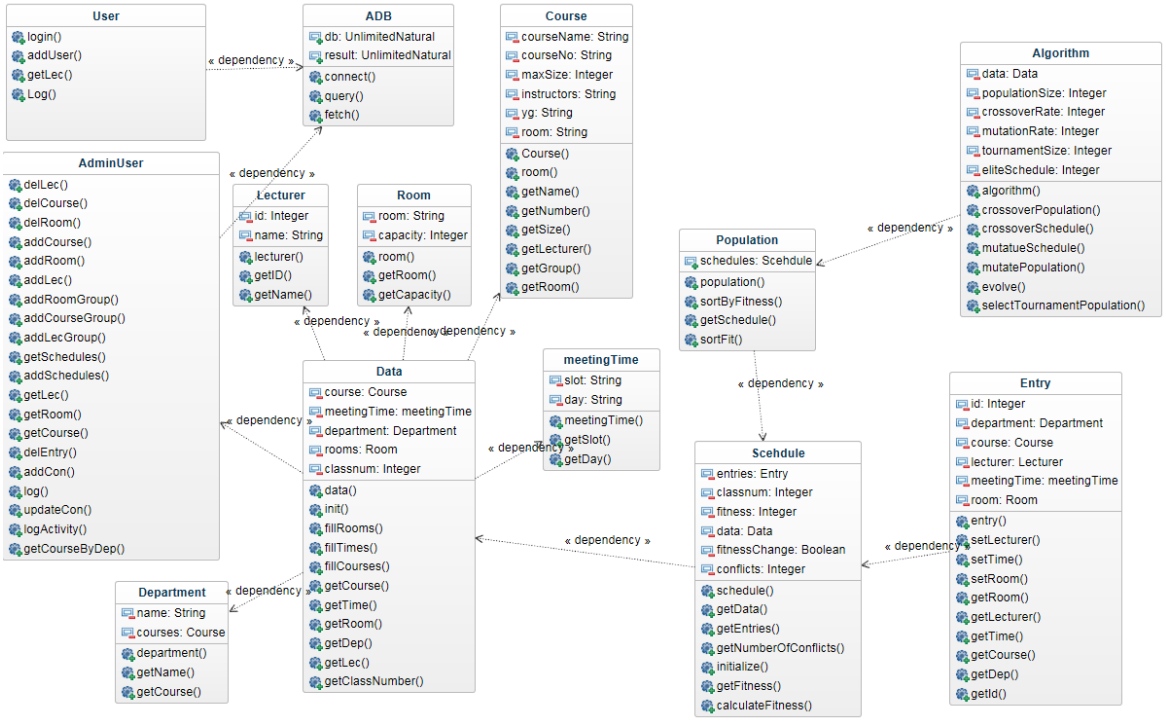


Figure 4.2: Class diagram for beehive scheduling application

### 4.2.3 Data Logic / Layer

The data layer is the third layer in the architecture. It is responsible for storing and retrieving data and handling SQL requests sent to it by the application layer. The information from the database is ultimately sent back to the user after processing.

The Beehive Scheduling Application uses a relational database to store data. It is convenient to use due to the relationships that exists between the room, course, lecturer, and schedule tables. It communicates with the application layer via SQL. SQL requests are received from the application layer and sent back in the form of results.

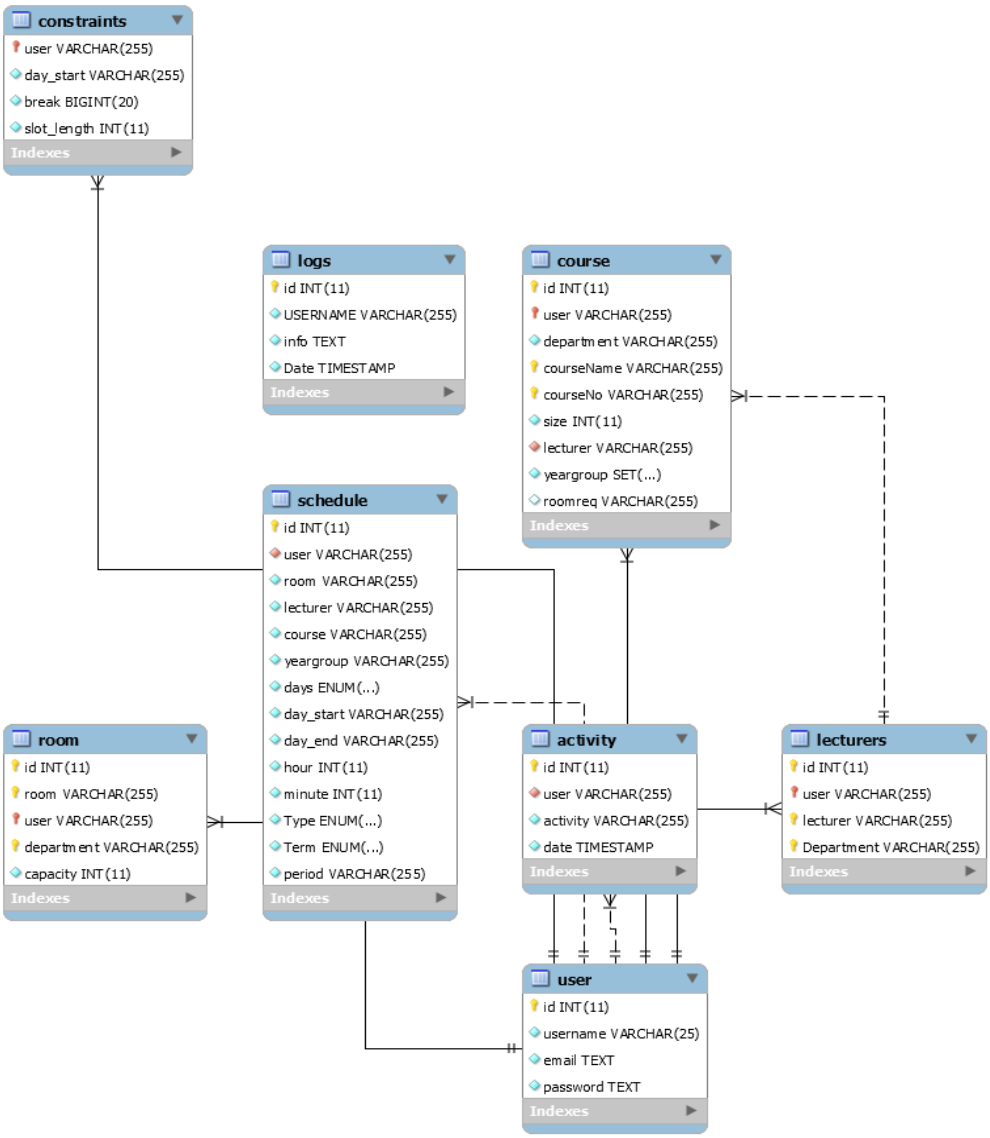


Figure 4.3: Enhanced entity relationship model for beehive scheduling application

## 4.3 External User Interface Requirements

This section looks at the individual interfaces of the system and describes the logical characteristics of each interface. (REBox, 2017). The focus of this section is on user interfaces in the interaction process.

The user interface is emphasized by vibrant colours categorized by shades of gold, teal, white and grey. Highlights and animations are used to guide the user. Alerts, dialogs, and string messages are used to give the user feedback on actions and serve as data input structures. The fonts used, Open Sans and Montserrat, are readable and easy on the eye. Buttons, Icons, Pagination, Links, and Sidebars are used to keep navigation and identification of elements simple. The application's ideal screen resolution is 1280 x 600 and would see elements distort if used beyond this. The main user interfaces of the software are below:

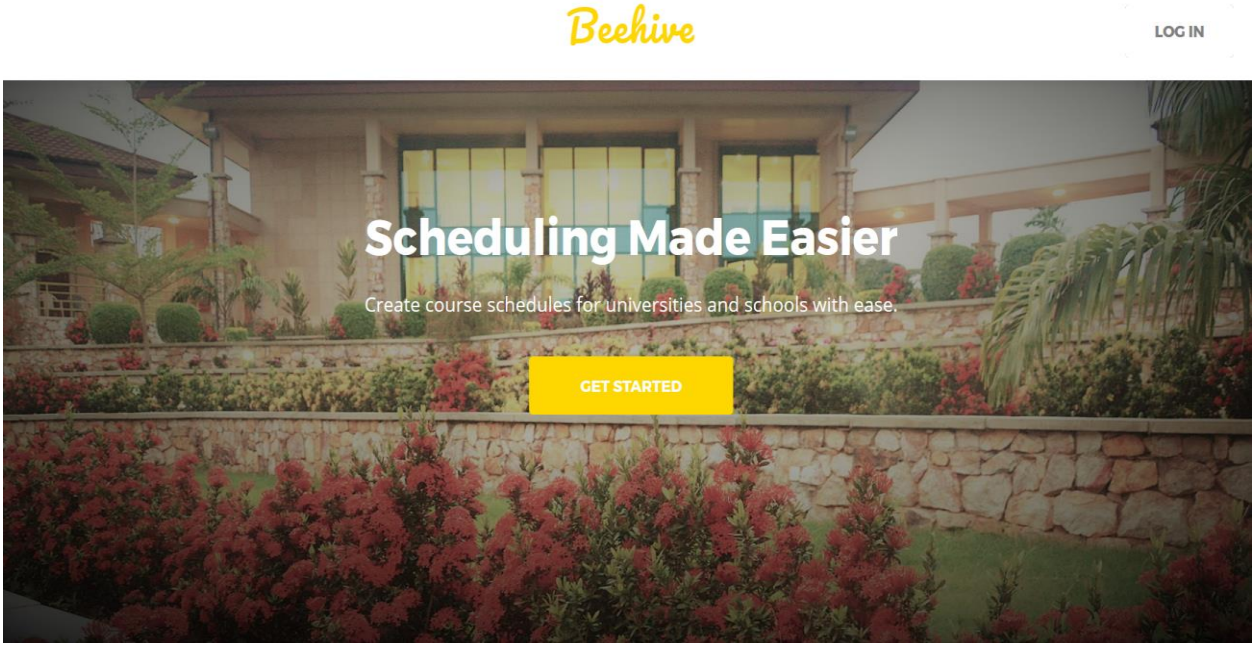


Figure 4.4: Landing or index interface of application

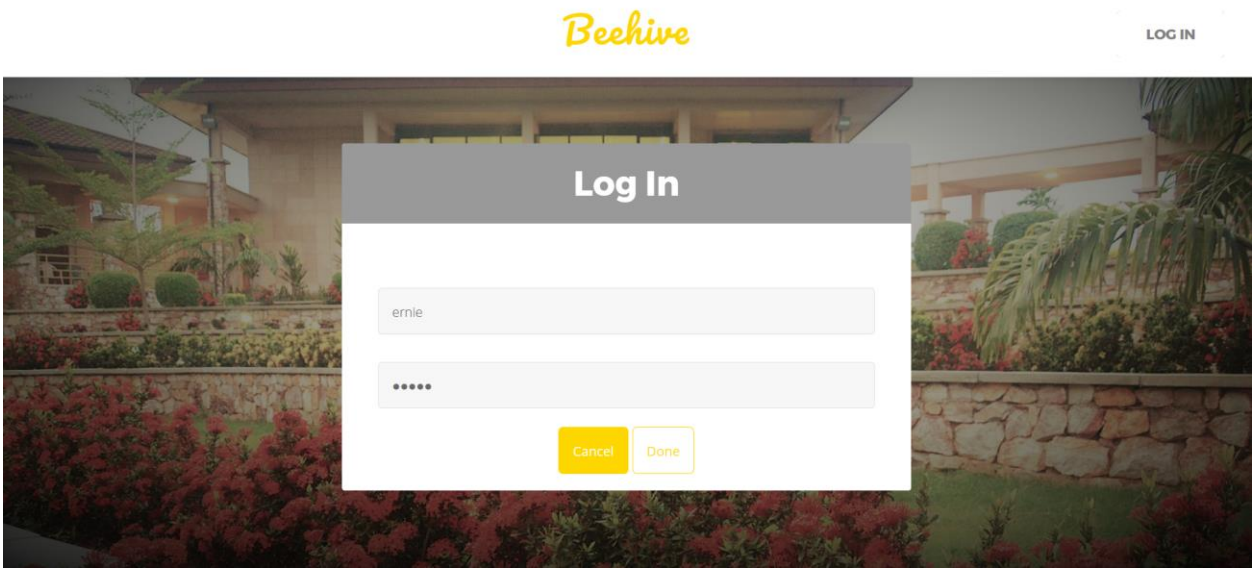


Figure 4.5: Log in dialog box on landing page

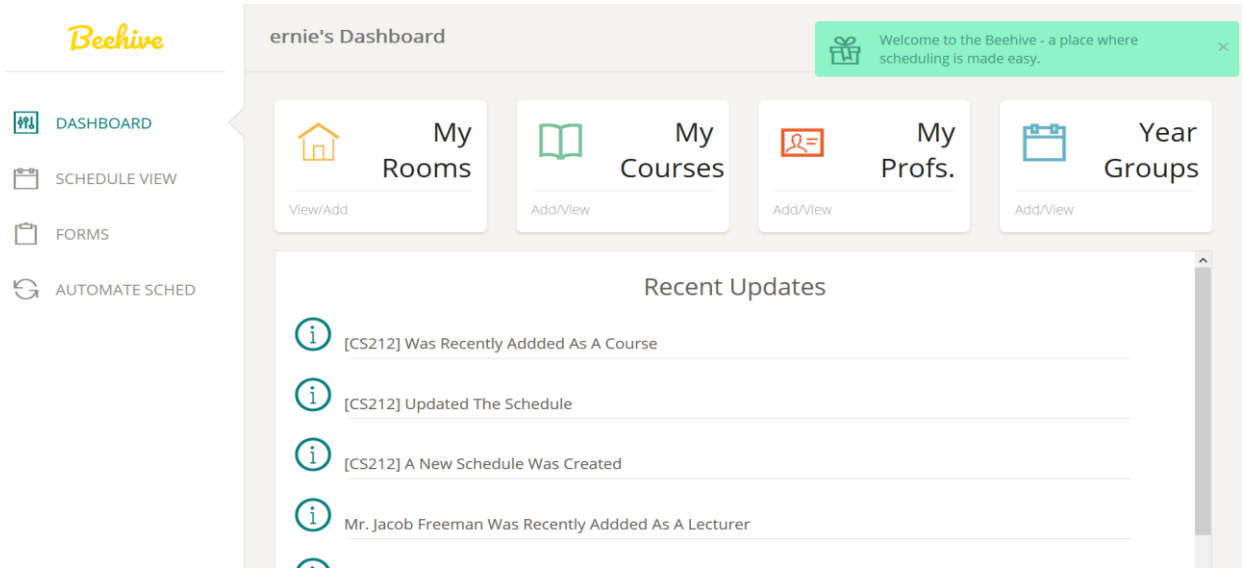


Figure 4.6: Dashboard of the scheduling application

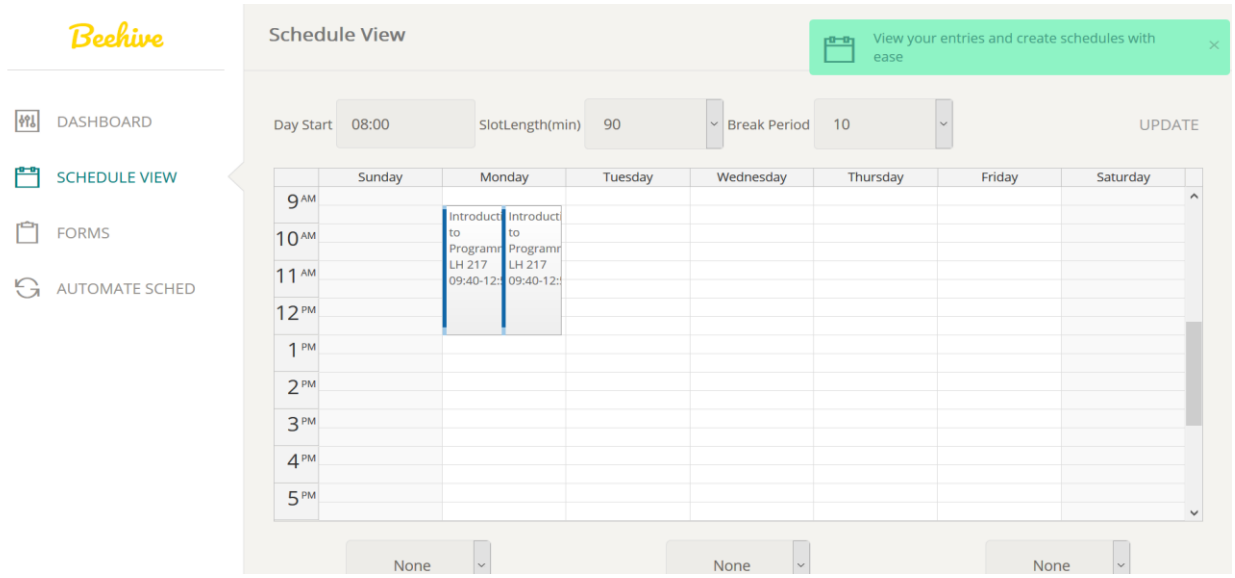


Figure 4.7: Schedule view interface of scheduling application

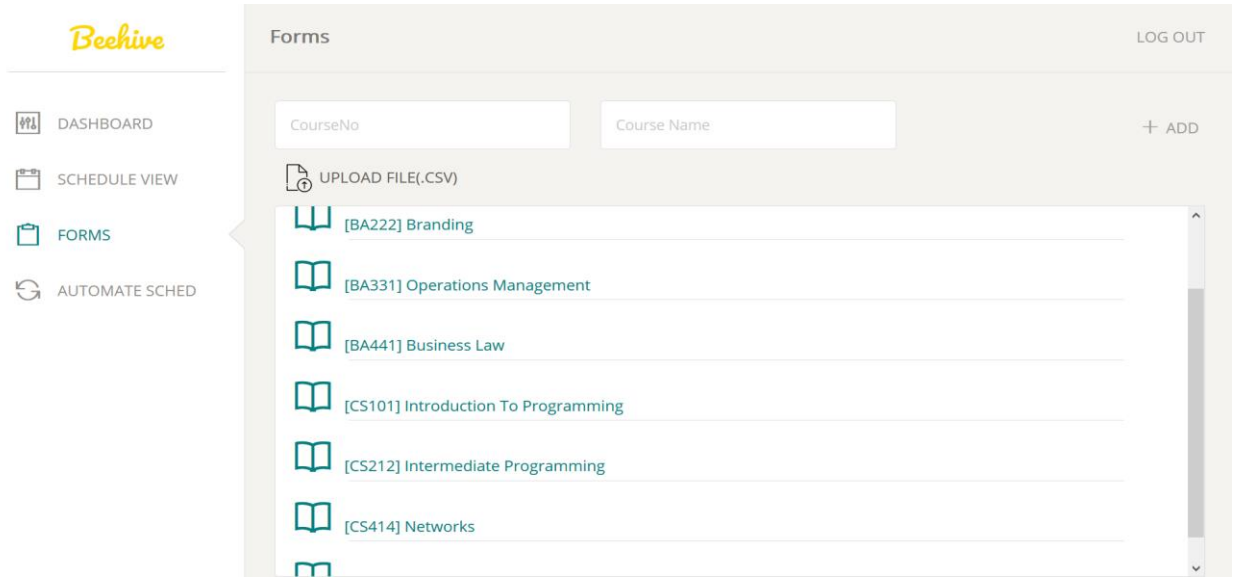


Figure 4.8: Forms (Data Addition) interface of scheduling application

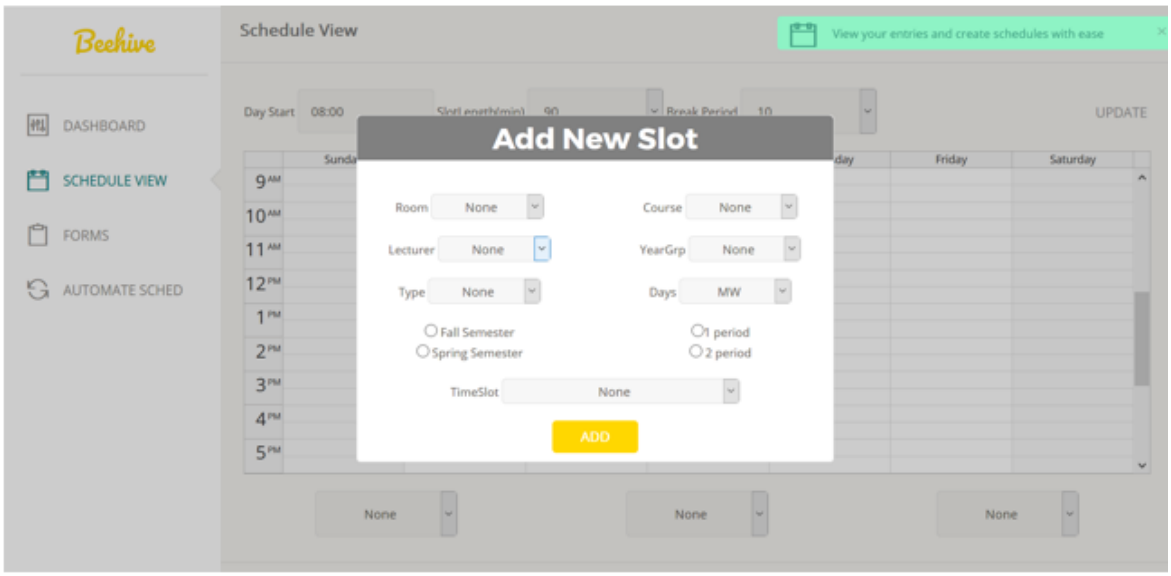


Figure 4.9: Schedule addition to the calendar on the schedule interface

# Chapter 5: Implementation

## 5.1 Overview

This chapter highlights the implementation of the Beehive Scheduling Application. It delves into the techniques and procedures used to build the application. It also looks at the implementation resources, tools and components used in the development of the Beehive Scheduling Application.

## 5.2 Implementation Resources

This section looks at the different languages, libraries, frameworks, Application Programming Interfaces (API's) and components used in the development of the application. The languages used include PHP, HTML, JavaScript, JSON, SQL, CSS, and AJAX. The application also makes use of various frameworks and libraries to simplify the development process.

### 5.2.1 Languages

**HTML**: The user interface of the application is built with HTML. HTML is used to describe the structure of the website and the different elements found on the application. Text, images, and other visuals are all defined in HTML files by elements. Elements defined by HTML are static but can be changed if necessary. The frontend files of the Beehive Application are HTML files with blocks of JavaScript code. All libraries and dependencies are defined in HTML files.

**CSS:** CSS is used to style and animate the elements in an HTML file. This is done by assigning IDs and classes to elements and applying styles to those classes or IDs. Beehive's sleek and minimalistic interface is heavily edited with CSS. The colour scheme of the interface is characterized by teal, gold, shades of grey and white. The design of

elements of the application is consistent and easy on the eye, highlighting event-driven elements to help the user navigate around the application.

**JavaScript:** JavaScript is used in the frontend of the application. It is needed to send requests to the server, implemented in PHP, and enforce the dynamic interactivity of HTML web elements. Beehive is heavily dependent on JavaScript as a lot of data is pulled from a database and dynamically displayed on the web interface. The event calendar that displays the schedule is fully dependent on JavaScript. All event-driven elements on the website like clicking and drag-and-drop are aided by JavaScript.

**AJAX:** The most important aspect of the Beehive's client-server architecture is the mode of communication between client and server. Beehive uses AJAX requests to send data and requests to the server side for processing and response. AJAX functions are implemented in JavaScript and require parameters like the destination URL and the data to send. The AJAX function has a success feature which receives the response from the server.

```
jQuery(document).ready(function ($) {
  $('.userSign').click(function (event) {
    event.preventDefault();
    var username=$('#sign_username').val();
    var password= $('#sign_password').val();
    var email= $('#sign_email').val();
    $.ajax({
      type: 'POST',
      url: 'ajax/projectAjax.php?cmd=1&password='+password+'&username='+username+'&email='+email,
      error: function () {
        // alert('error, failed to get id');
      },
      dataType: 'json',
      success: function (response) {
        if(response.message=="false")
        {
          $(".prompt").html("Please Make Sure All Fields Are Filled");
        }
        else if(response.message=="fail")
        {
          $(".prompt").html("Please Choose Another Username");
        }
        else
        {
          $(".prompt").html("Account Successfully Created");
        }
      },
    });
  });
});
```

Figure 5.1: Screenshot of AJAX function

**JSON:** The server responds to the client using JSON. JSON encodes the data in a form which both languages can interpret and decode. Beehive uses JSON to transfers arrays, objects, strings, integers, and any permitted data type from the server to the Client. The client receives the JSON message via the AJAX function.

{"message":"Schedule was Successfully generated","result":"0"}

Figure 5.2: Screenshot of JSON reply

**PHP:** Beehive's server side is built with PHP. As seen in Chapter 4, the server side contains the application's business and data management logic. PHP is used to build classes and functions necessary for receiving and responding to requests from the client and communicating with the data storage entity. The server sends insertion, deletion, updating or fetching queries to the database and receives results on success.

```
class users extends adb{

    function addUser($username,$password,$email){
        $strQuery="insert into user set
                        USERNAME='$username',
                        EMAIL='$email',
                        PASSWORD='$password'";
        return $this->query($strQuery);
    }

    function login($username,$password)
    {
        $strQuery= "SELECT username,user.id,day_start,break,slot_length FROM user LEFT JOIN constraints ON user.
        return $this->query($strQuery);
    }
```

Figure 5.3: Screenshot of PHP code

**SQL:** SQL commands are used to query data in the relational database. SQL queries are sent to the database manager for execution. The database manager is PHPMyAdmin. PHPMyAdmin is an interactive graphical user interface which simplifies the management of the database. Result sets are sent back to the server from the database manager. The

database is mainly used for storing data, however it can execute queries to perform functions like deletion, fetching, and insertion.

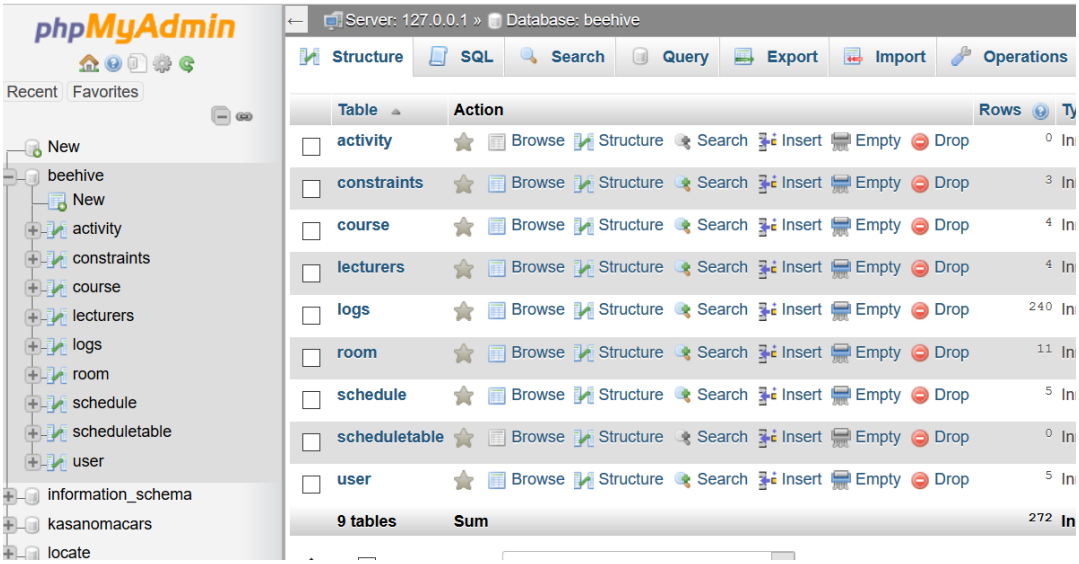


Figure 5.4: Screenshot of database manager

### 5.2.2 Frameworks

**Bootstrap:** Beehive's frontend is built on Bootstrap, an open source framework built on JavaScript, HTML, and CSS for developing responsive and mobile friendly websites. Bootstrap simplifies the development process by providing documentation for custom HTML elements and JavaScript plugins.

- **Paper Dashboard and Intensify**: Beehive's admin pages and index pages are built on two free licensed bootstrap templates, Paper Dashboard and Intensify respectively.

### 5.2.3 Libraries

**DayPilot:** DayPilot is an open source JavaScript library which dynamically creates an event calendar on a website. Beehive uses this to display the scheduled classes.

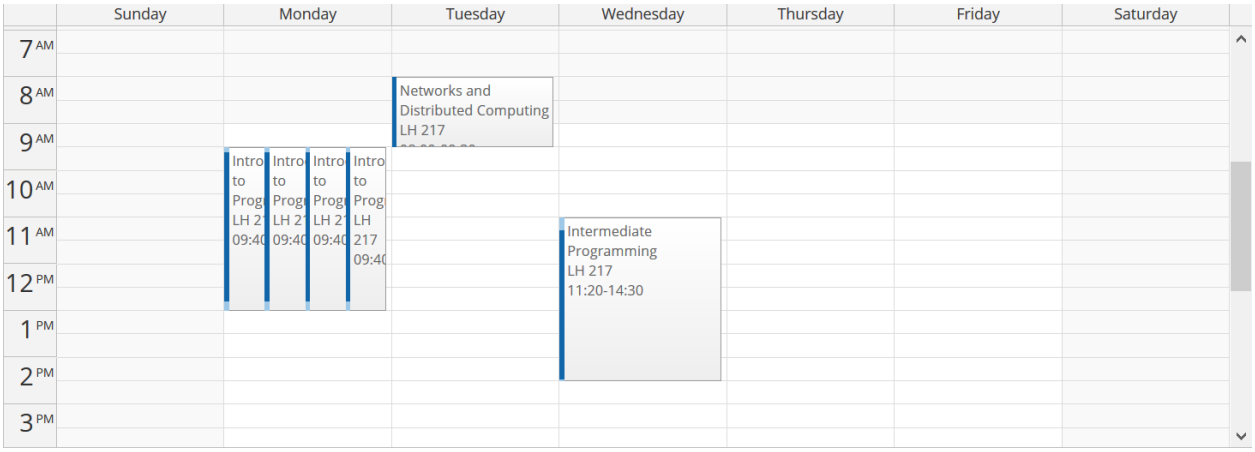


Figure 5.5: Screenshot of DayPilot in use on Beehive

## 5.3 Implementation Techniques

### 5.3.1 Login and Sign Up

User Accounts allows for the personalization of Beehive. A user can create an account by entering his email, username, and password in a modal window that appears upon clicking the sign-up button. Once done, the user is notified on success or failure. On success, the user can then proceed to log in into the application's admin panel. The admin panel is basically the part of the application for data manipulation and schedule generation. The user requires username and password to be authenticated to log in to the application. The user logs in via a modal window like that of the sign-up modal. When logged in, the user is redirected to his dashboard.

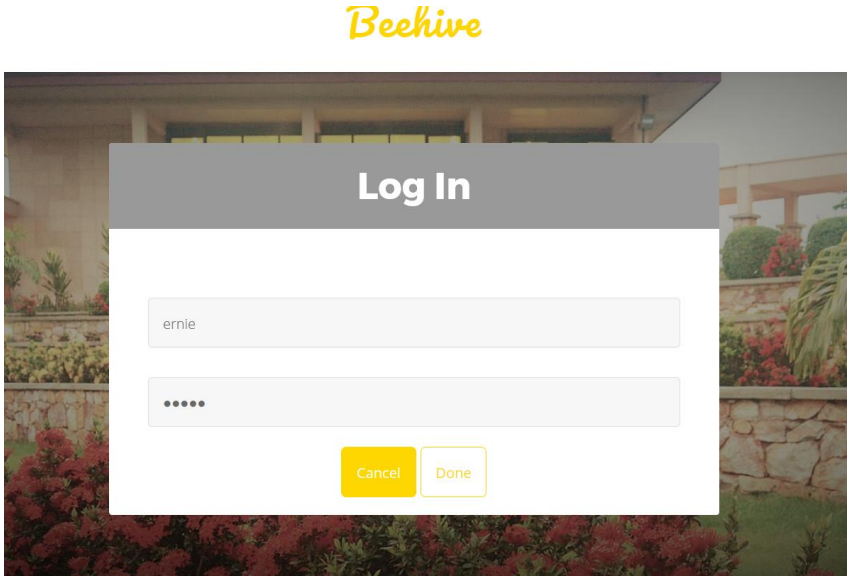


Figure 5.6: Screenshot of login/sign up window

### 5.3.2 Data Input

For a new user, the application would have no data and hence the user would be required to provide data for class and schedule creation. Data input can be done on the forms page. A user can perform the following:

- Enter the desired name and course number of the course.
- Enter the desired name and department of the room.
- Enter the desired name of the lecturer and his department.
- Add data via a Comma Separated File(CSV).

When complete, the user clicks done and the new entry can be seen in the panel below the input fields. On success or failure, the user would be notified accordingly. The pagination below the panels is used to switch between the data input options (rooms, lecturers, courses, classes/entries).

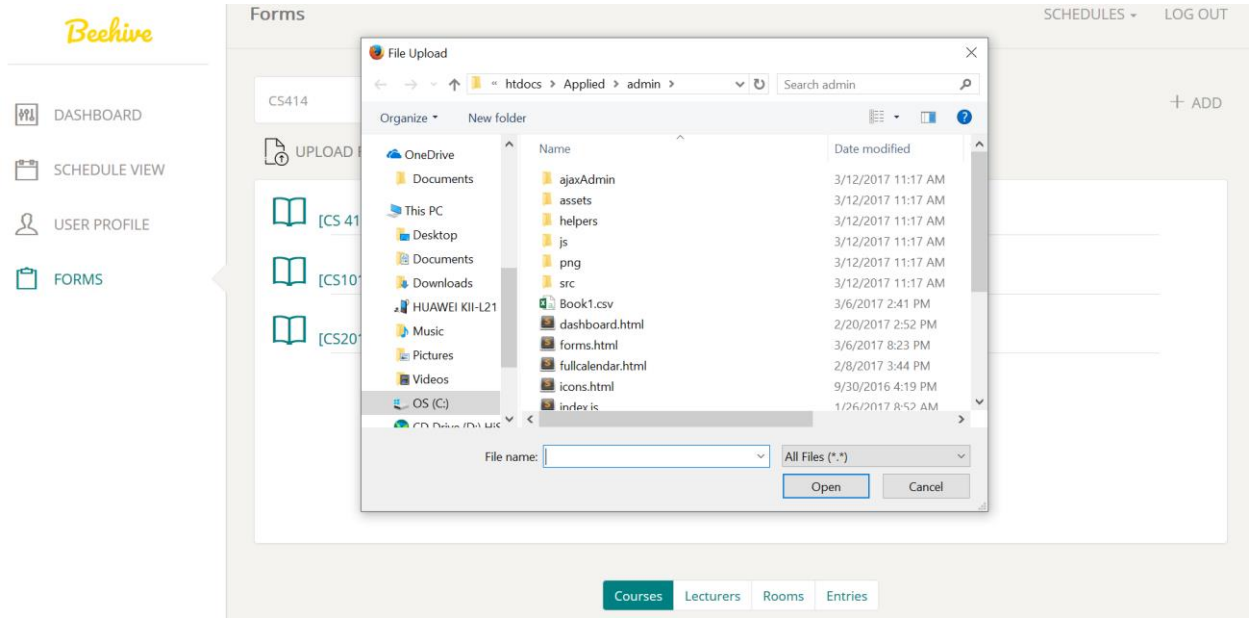


Figure 5.7: Screenshot of data input via input fields and file upload

### 5.3.3 Schedule Display

Schedules are displayed on an event calendar found on the schedule view page. Classes are displayed as boxes on the time slots on the event calendar with the respective information of that class.

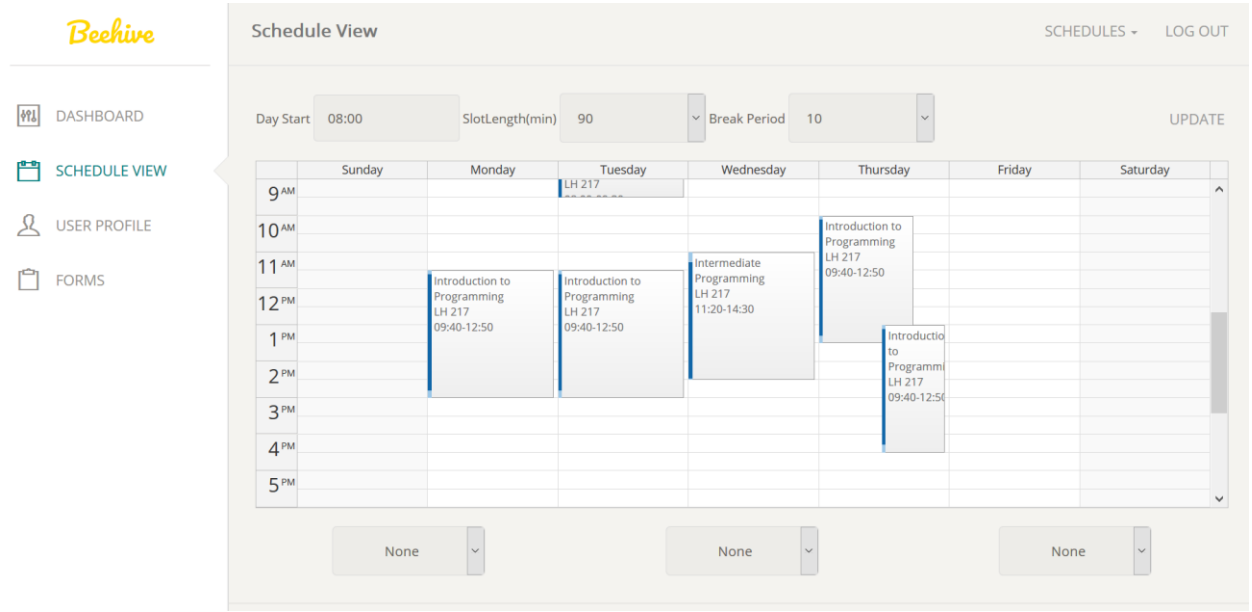


Figure 5.8: Screenshot of schedule view

### 5.3.4 Schedule Creation

Classes can be added to the event calendar by simply clicking on a day. When clicked, a modal pops up. The modal fetches data entered into the database and makes them accessible to the user by putting them in drop downs. Other parameters to describe the class like period, semester, year group, time slot, days taken are also available on the modal. When satisfied, the user simple clicks the add button and the entry is inserted into the calendar and stored in the database. For a new user, requirements like the time the day starts, slot length in minutes and break period between each course must be fulfilled before being able to add an entry. If a new user tries to add an entry without completing these, the user would be prompted to do so before being allowed to create a schedule.

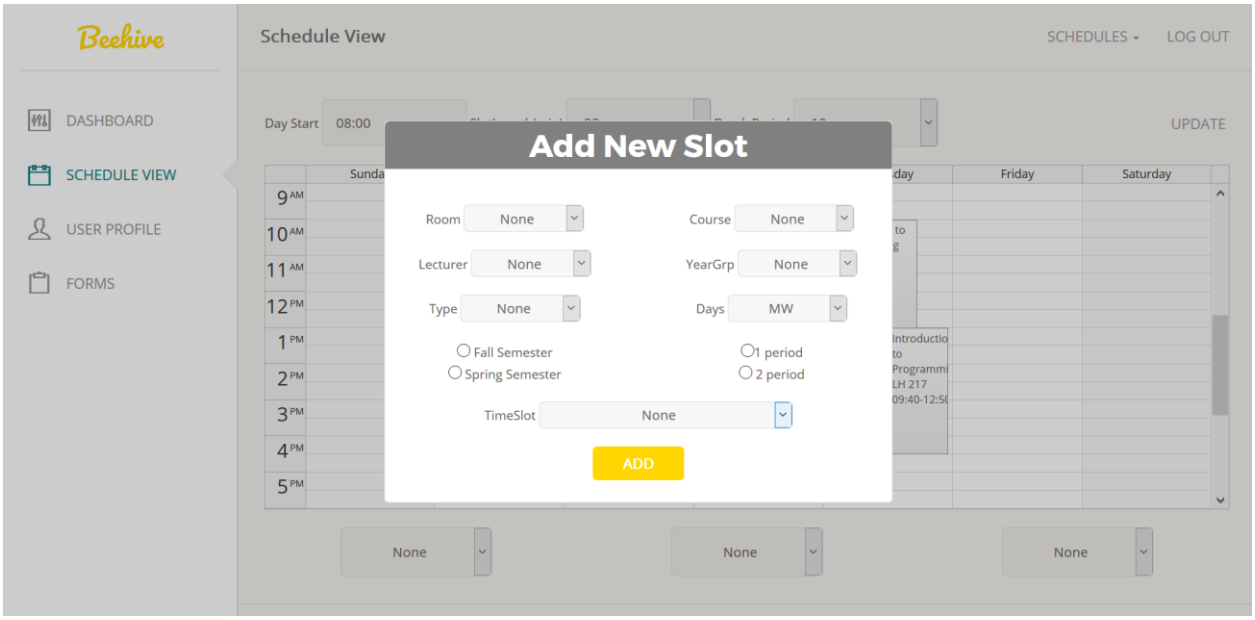


Figure 5.9: Screenshot of schedule creation

#### 5.3.4.1 Genetic Algorithm

The Genetic Algorithm stands as the core of Beehive's functionality. It is responsible for generating an ideal schedule based on the data and constraints presented to it. Genetic Algorithms are built on the concept the Darwinian Theory of "Survival of the Fittest" and

hence prioritize three main things: Variation, Heredity, and Selection. It is an evolutionary algorithm and hence iterates over a population to evolve into an ideal solution (Neville & Sibley, 2003). Beehive's Genetic Algorithm was built with the aid of materials from ZA Software Development Tutorials, The Coding Train, Tutorialspoint and Code Project. The algorithm is built on PHP and hence resides in the server side of the application. The algorithm's logic was built using ten classes which represent objects necessary for course scheduling like classes, schedules, and rooms.

The algorithm starts off with a data object and a population object. The data object pulls all the necessary data needed to create a schedule from Beehive's database. A population object then uses this data to create schedules with randomized classes. The randomization of schedules ensures that there is variation in the population and hence there is a high tendency for the occurrence of conflicts.

```
/**
 * [[Constructor of the class]]
 * @param [[Integer]] $size [[Size of the population]]
 * @param [[Array]] $data [[Array of Data]]
 */
function population($size, $data)
{
  $this->schedules =  array();
  for($i=0;$i< $size;$i++)
  {
    $sch = new schedule($data);
    array_push($this->schedules,$sch);
  }
}
```

Figure 5.10: Population class constructor

Each schedule has a fitness function which calculates how fit the schedule is. The fitness of the schedules is calculated based on how many conflicts it has based on the constraints specified. The fitness function can be found below.

$$Fitness = \left(\frac{1}{number\ of\ conflicts + 1}\right)^4$$

The population sorts the schedules by fitness in descending order. If the elite schedule has a fitness of 1, then that schedule is conflict-free and is selected as the solution. If not, the population goes through a process called crossover. Selection is done when crossing over. In crossing over, a whole new generation of schedules is generated from two parents, the elite schedule of the current population and any two other random schedules selected based on randomization. The purpose of this is to keep on creating a new generation of schedules which would eventually converge at an optimal schedule.

```
function crossoverPopulation($population)
{
 $crossoverpopulation = new population(count($population->getSchedule()),$this->data);
 for($i =0; $i <  $this->eliteSch ; $i++){
  $crossoverpopulation->getSchedule()[$i]=$population->getSchedule()[$i];
 }

 for($i =$this->eliteSch; $i < count($population->getSchedule())  ; $i++)
 {
  if ( $this->crossoverRate > (mt_rand()/mt_getrandmax()) )
  {
   $schedule1= $this->selectTournamentPopulation($population)->sortByFitness()->getSchedule()[0];
   $schedule2=$this->selectTournamentPopulation($population)->sortByFitness()->getSchedule()[0];
   $crossoverpopulation->getSchedule()[$i] = $this->crossoverSchedule($schedule1,$schedule2);
  }
  else
  {
   $crossoverpopulation->getSchedule()[$i]=$population->getSchedule()[$i];
  }
 }
 return $crossoverpopulation;
}
```

Figure 5.11: Crossover method in algorithm class

Every time instance of a crossover, comes with a chance of mutation. Mutation replaces all schedules apart from the elite's schedule with new randomized schedules different from those within the population. The purpose of this is to ensure that variation is maintained in the population as the algorithm depends on a varied population. Mutation only occurs if a picked random number falls below the mutation rate specified.

```
/**
 * [[Function to mutate population]]
 * @param  [[population]] $population [[population object]]
 * @return [[population]] [[mutated population object]]
 */
function mutatePopulation($population)
{
 $mpopulation = new population(count($population->getSchedule()),$this->data);
 $schedules =  $mpopulation->getSchedule();
 for($i =0; $i <  $this->eliteSch ; $i++){
  $schedules[$i]=$population->getSchedule()[$i];
 }
 for($i =$this->eliteSch; $i <  count($population->getSchedule()) ; $i++){
  $schedules[$i]=$this->mutateSchedule($population->getSchedule()[$i]);
 }
 return $mpopulation;
}
```

Figure 5.12: Mutation method in algorithm class

After a crossover, the fitness of the elite schedule is checked again to see whether it is conflict free. If not, a new generation is created. This process goes on and on until a conflict free schedule is found. The code and algorithm listings can be found at the end of the document.

```
$da = new data();
$algo = new algorithm($da);
$pop = new population(12,$da);
$pop =$pop->sortByFitness();
$generation =0;
$successful = false;
while($pop->getSchedule()[0]->getFitness()!=1.0)
{
    $pop = $algo->evolve($pop)->sortByFitness();
}
//print_r($pop->getSchedule()[0]->getEntries());
$successful = true;
if($successful == true)
    echo '{"message":"true"}';
else
    echo '{"message":"false"}';
```

Figure 5.13: Driver of algorithm

# Chapter 6: Testing and Results

## 6.1 Overview

This chapter looks at the various methods used to test the functionalities of the Beehive Scheduling Application and respective results. Unit testing of functions, component testing, system, and user testing will be carried out to verify and validate the requirements of the application. A Dell Inspiron 13 Core I7 was used in testing and debugging the application.

## 6.2 Unit Testing

Unit testing is an important aspect of programming development that helps to find errors and correct them to ensure the proper functioning of the application. (Sommerville, 2011) The functions was tested by passing various inputs to see whether they will return the expected result. Data management functions residing in the server were tested using PHPUnit. To test the AJAX functions, the AJAX URL's were passed into browser to test whether the proper JSON response was sent from the server. Postman, a GUI for constructing requests and reading responses, was also used to help test the AJAX functions. The table below highlights some of the main functions that were tested using unit testing.

Table 6.1: Summary of Unit Testing

| Unit Test Point | Testing | Result |
|---|---|---|
| **addUser method in User Class** | Write code to Create an object of the class and call the method with test inputs and compare output in database<br>Test input= username= assigned random bytes method, Email=john@john.com, password=blah | Result: Data was inserted into the database |
| **getUser method in User Class** | Write code to Create an object of the class and call the method to get result set of users and compare output in database | Result= array of results |

| | | |
|---|---|---|
| **Connect method in adb class** | Write code to Create an object of the class and call the method to connect to database. | Result= connected |
| **Query method in adb class** | Write code to Create an object of the class and call the method query to return true if successful. | Result= true |
| **Fetch method in adb class** | Write code to Create an object of the class and call the method to return a result set of the query. | Result= array of results |
| **delLec method in User Class** | Write code to Create an object of the class and call the method query to return true if successful.<br>Test input: id of user. | Result= lecturer deleted from database |
| **getRooms method in User Class** | Write code to Create an object of the class and call the method query to return true if successful. | Result= array of results |
| **updateCon method in User Class** | Write code to Create an object of the class and call the method query to return true if successful.<br>Test input: username='ernie', day_start='8:00', break='10', slot_length='90' | Result= row with username ernie is updated in database |

```php
<?php
include_once("..\users.php");

class testAdb extends PHPUnit_Framework_TestCase
{
    public function testAddUser()
    {
        // generate test
        $strTestUsername=random_bytes(10);
        $obj=new users();

        $this->assertEquals(true,
        $obj->addUser(
            $strTestUsername,// username sername
            "john@gmail.com",   //email
            "johnKufuor",           //password
            ));

        $this->assertEquals(true,$obj->query("select * from users where username='$strTestUsername'"));
        //count the number of fields
        $this->assertCount(7,$obj->fetch());
    }
```

Figure 6.1: Snippet of PHPunit test class.

## 6.3 Component Testing

Components testing looks at the testing and debugging of system components, functions or classes interacting to perform a common function. (Sommerville, 2011) Beehive's main components were tested by viewing results from the user interface. The main components of the application are logging in, signing up, viewing data, deleting data, inserting data, and creating classes.

## 6.4 User Testing

User testing is one of the most important aspects of software development. Here, user or potential users of the system try out the application and give feedback to the developer. Beehive is a course scheduling application and hence it was tested by Ashesi's academic registry for feedback and error detection. The Registry pointed out certain aspects of course scheduling that the application did not capture. Consequently, the application was updated accordingly.

# Chapter 7: Conclusion and Recommendations

## 7.1 Challenges

When developing the Beehive Scheduling Application, these challenges were encountered.

1. Defining the scope of the constraints and variables: Hardly do institutions share the same schedule structures and models and hence they differ in structure. This made it difficult to consider which variables to include in development.
2. Finding an algorithm and getting it to work: This was the biggest challenge, as deciding on which algorithm to use and making it work with data was a big worry.
3. Working within the time constraints: Although familiar with the programming languages, a lot of the implementations were new and took a lot of time to build.

## 7.2 Conclusion

The aim of this project was to find a solution to the classical course scheduling problem. A lot of systems and algorithms were considered. Common gap was pointed out that no system is 100% efficient and each output is susceptible to change. Beehive Scheduling Application was introduced as a solution to this problem. Beehive offered both manual and automated scheduling in one software and this simplified the course scheduling process. It can be said that Beehive is an easy to use general course scheduling software to be used by institutions. The main limitation with Beehive is its inability to satisfy the constraints for every single institution as the scheduling criteria for institutions differ as it did not fully satisfy Ashesi's constraints. However, Beehive's manual accessibility helps bridge the gaps automation leaves. The algorithm also takes some time to generate a solution as it goes through several generations.

For future works, Beehive can be improved upon to satisfy more constraints necessary per an institution. The algorithm can be optimized further to perform better, have

a better run time, and provide better results. It can also be extended to provide more functionality where necessary.

# References


Agbenowosi, J. F. (2014). *Automated Course Scheduler For Ashesi.* Berekuso: Ashesi University College.

Castrillón, O. (2015). Cognitive rhythms and evolutionary algorithms in university timetables scheduling. RMTA, 22(1), 135. Retrieved from: http://dx.doi.org/10.15517/rmta.v22i1.17559.

Dahiya, S. (2015). *Course Scheduling With Preference Optimization.* Pennsylvania: The Pennsylvania State University.

Donkor, G. (2014). *A Process for Automated Class Scheduling at Ashesi.* Berekuso: Ashesi University College.

Hamalainen, W. (November 6, 2006). Class NP, NP-complete, and NP-hard problems.Finland: http://www.uef.fi.

Hinkin, T. R., & Thompson, G. (2002). SchedulExpert: Scheduling Courses in the Cornell University School of Hotel Administration. *The Scholarly Commons*, 57.

Janković, M. (2008, January 22). *Making a Class Schedule Using a Genetic Algorithm*. Retrieved from: https://www.codeproject.com/Articles/23111/Making-a-Class-Schedule-Using-a-Genetic-Algorithm.

Karami, A., & Hasanzadeh, M. (2012). University course timetabling using a new hybrid genetic algorithm. *2012 2Nd International Econference On Computer And Knowledge Engineering (ICCKE)*. Retrieved from: http://dx.doi.org/10.1109/iccke.2012.6395368.

Mingo, S. (2000). A Brief History of Automation – SEMCI: A Paperless Future. *Insurance Journal West Magazine*.

Neville, M., & Sibley, A. (2003). Developing a Generic Genetic Algorithm. *ACM SIGAda Ada Letters* , 45-52.

Papafio, W. N. (2015). *Ashesi Course Scheduling.* Berekuso: Ashesi University College.

REBox. (2017, March 21). *External Interface Requirements*. Retrieved from: http://isg.inescid.pt/REBox/WiegersTemplate@17.aspx?page=4ExternalInterfaceRequirements.

Siddharth, D. (2015). *Course Scheduling with preference optimization*. Pennsylvania: The Pennsylvania State University.

Sommerville, I. (2011). *Software Engineering.* Boston: Addison-Wesley.

The Coding Train (2016, July 29). *Genetic Algorithms - The Nature of Code* [Video file]. Retrieved from https://www.youtube.com/watch?v=9zfeTw-uFCw&list=PLRqwX-V7Uu6bJM3VgzjNV5YxVxUwzALHV.

Tutorialspoint. (2016). *Genetic Algorithms - Introduction.* Retrieved from: https://www.tutorialspoint.com/genetic_algorithms/index.htm.

United Nations. (2003). *Population,Education and Development: The Concise Report.* New York: United Nations Publication.

Wasfy, A., & Aloul, F. A. (2006). *Solving the University Class Scheduling Problem Using Advanced ILT Techniques*. Sharjah: American University of Sharjah publications.

ZA Software Development Tutorials (2016, September 15). *Genetic Algorithms Tutorial 04 - Class Scheduling JAVA Application* [Video file]. Retrieved from https://www.youtube.com/watch?v=cn1JyZvV5YA&t=389s.