Class Planning and Management System

A dissertation submitted in partial fulfilment of

the requirements for the degree of

BACHELOR OF SCIENCE in Computer Science

in

The Queen's University of Belfast

by

Tiarnán Duffy

11/04/2024

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE

CSC3002 – COMPUTER SCIENCE PROJECT

Dissertation Cover Sheet

A signed and completed cover sheet must accompany the submission of the Software Engineering dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

|  |  |  |  |
| --- | --- | --- | --- |
| Student Name: | Tiarnán Duffy | Student Number: | 40293936 |
| Project Title: | Class Planning and Management System | | |
| Supervisor: | Barry McCollum | | |

Declaration of Academic Integrity

Before submitting your dissertation please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

**By submitting your dissertation you declare that you have completed the tutorial on plagiarism at** [**http://www.qub.ac.uk/cite2write/introduction5.html**](http://www.qub.ac.uk/cite2write/introduction5.html) **and are aware that it is an academic offence to plagiarise. You declare that the submission is your own original work. No part of it has been submitted for any other assignment and you have acknowledged all written and electronic sources used.**

1. If selected as an exemplar, I agree to allow my dissertation to be used as a sample for future students. (Please delete this if you do not agree.)

|  |  |  |  |
| --- | --- | --- | --- |
| *Student’s signature:* | Tiarnán Duffy | 11/04/2024 |  |

A close up of a writing

Description automatically generated

Acknowledgements

I'd like to thank my supervisor, Barry McCollum, for his help and guidance on the project over the year. I would also like to thank my family for their support, especially my Dad and sister for their help over the entire 4 years of studying Computer Science at Queen's.

Abstract

A system that focuses on the creating of timetables for a university by scheduling courses with their modules and classes. The system allows the user to set various constraints and optimisation options on the schedule which are taken into account when the Scheduling Engine operates. The engine uses an intelligent algorithm to ensure the modifications and constraints are applied and followed while completing the schedule efficiently. Also, the system allows the user to manage the data of the university's various entities and facilitates the user to access the system as a student or lecturer to perform tasks specific to them.

*Source code is stored in two Gitlab repositories, one for frontend* [1] *and one for backend* [2]*.*

Table of Contents

[1.0 Introduction and Problem Area 5](#_Toc163755030)

[1.1 Scheduling Timetables for University 5](#_Toc163755031)

[1.2 Understanding the Problem 5](#_Toc163755032)

[1.3 Proposed Solution - 5 Key Features 6](#_Toc163755033)

[2.0 System Requirements and Specification 8](#_Toc163755034)

[2.1 Web Application 8](#_Toc163755035)

[2.2 Optimisation Engine 11](#_Toc163755036)

[3.0 Design 14](#_Toc163755037)

[3.1 Architectural Description 14](#_Toc163755038)

[3.2 User Interface Design 15](#_Toc163755039)

[3.3 Software System Design Component Breakdown 19](#_Toc163755040)

[4.0 Implementation 21](#_Toc163755041)

[4.1 Choice of Implementation Language and Environment 21](#_Toc163755042)

[4.2 Use of Software Libraries 22](#_Toc163755043)

[4.3 Key Implementation Decisions 30](#_Toc163755044)

[4.4 Schedule Algorithm 32](#_Toc163755045)

[5.0 Testing 39](#_Toc163755046)

[5.1 Postman System Testing 39](#_Toc163755047)

[5.2 System Testing 42](#_Toc163755048)

[6.0 System Evaluation and Experimental Results 47](#_Toc163755049)

[7.0 Appendices 49](#_Toc163755050)

[Bibliography 51](#_Toc163755051)

1. Introduction and Problem Area
   1. Scheduling Timetables for University

For students and lecturers alike, the timetable of their university week could be considered the single most important intangible thing related to their experience. From the most basic and obvious perspective, for lecturers to be able to teach their students, and for students to be able to learn from their lecturers, both parties need to have availability to do this. Not just that both have the free time at a specific, decided upon slot; but that there is a decided upon room to do this in, that there is enough capacity in said room, that there is the correct facility available to do this in, that all students in the module are also free. Indeed, before anything else shapes the university experience, a proper schedule must be established.

However, to get the most effective learning, teaching, and overall enjoyment for both, schedules must be able to be modified and optimised to ensure a fair and balanced timetable.

For one example, when planning a schedule, those doing so might want to ensure that classes are spread out throughout the week and all not on the same day. It is essential that students have a good work to life balance. Having too many classes on at once, might cause a build-up of assignments and lecturers to catch up on, which might have them feel totally overwhelmed by study and life demands [1] . This can lead to Burnout. Burnout is a mental exhaustion accompanied by decreased motivation, lowered performance [2] and therefore universities will want to avoid this. With the Class Planning and Management System there will be modifications in place to prevent this from happening, preventing the problem of burnout.

Elsewhere, lecturers might prefer to schedule all their classes towards the end of the day, or perhaps they may want different types of classes to be scheduled in a specific order if it suits their learning curriculum better. They may want to schedule an advisory session early in the week, and then want their students to apply this knowledge in a lab session. But this lab session requires computers.

There are many different constraints placed on a single module or course that affects its ability to be scheduled.

* 1. Understanding the Problem

So to take all applied modifications into account while adhering to the hard constraints of:

* Lecturer Availability - (Slots taken for lectures other classes. Lectures also can access the system and reserve slots in which they are unavailable)
* Room Availability - (Both capacity wise, and facility wise)
* Student Availability - (No conflict/overlap with other classes, alike lecturers)

will be an issue. It is going to be a rare occurrence that courses will consistently schedule with all these slots free, while complying with the modifications applied. There will certainly be times, where suitable rooms are fully booked, or all students aren't available.

Herein, the problem arises, there must be an intelligent optimisation algorithm in place which allows these hard constraints to be heuristically softened, to make proper scheduling feasible.

* 1. Proposed Solution - 5 Key Features

**1 - Admin Management Role**

* Have the ability and permissions to Create, Edit, and Delete:
  + Lecturers and Students.
  + Courses, the Course's Modules, and the Module's Classes.
  + Buildings and their Rooms.

**2 - Create Schedule**

* Set Optimisations.
* Pre-set soft constraints that the algorithm will take into account when scheduling.

**3 - Algorithm**

* Apply the optimisations on schedule.
* Start schedule engine.
* Store succeeded and failed classes.
* Store reasons for why classes failed.
* Store suggestions on what constraints to soften to make the schedule succeed.

**4 - Timetable Outcome**

* Display succeeded and failed modules.
* Display succeeded and failed classes for a selected module.
* Display reasons for why classes failed.
* Display suggestions on how to use heuristics to soften the constraints to make schedule work.
* Allow admins to apply the suggestions rather than manually go back and apply them on 'create'.
* Reschedule button with newly applied constraints.

**5 - Lecturer and Student User Interface**

* Lecturers reserve slots. Their classes will not be scheduled at these slots.
* Students can select modules from their course year.
* Both can view their timetable.

1. System Requirements and Specification

The system requirements and specification can be split into two main parts.

1. This project requires a **web application** with an intuitive user interface that provides Admins the capacity to create the data that will be used in the system and stored in a robust data management layer. *Lecturers* and *Students* will be created, in which they teach or study *Modules* which are a part of a *Course* and have different *Classes*. *Rooms* are needed to hold these classes, and they are a part of a *Building*.

Lecturers need to be able to reserve slots and view the modules they teach, and Students need to select modules.

1. The project also requires an **intelligent optimisation engine** that will schedule a course, as effectively and efficiently as possible, while catering for constraints and abiding by optimisation modifications applied by the admin when creating the schedule.

The engine must seamlessly interface with the data management layer to hold information about the outcomes of the schedule, so that it can be delivered back to the admin, and if successful the time slots of the users must be updated.

To get the best understanding of the system requirements, Part 1 will be a Requirements Definition via User Stories, and Part 2 will be a Software Specification.

* 1. Web Application

Functional Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature ID | Feature | User Story | Requirement Description | Priority | ID |
| F1 | Select User | As a user I want to be able to select the type of user I am to login | User will select either Student, Lecturer, or Admin | High | F1-REQ1 |
| As a student I want to access the system as my specific user | Student will login with Local Storage setting their User ID | High | F1-REQ2 |
| As a lecturer I want to access the system as my specific user | Lecturer will login with Local Storage setting their User ID | High | F1-REQ3 |
| F2 | Navigation and Permissions | As an admin I want access to pages specific to my permissions | The side nav links will allow the admin access to their CRUD and Create Timetable responsibilities | Medium | F2-REQ1 |
| As a student I want access to pages specific to my permissions | The side nav links will be 'Home', 'Select Modules' and 'View Timetables' | Medium | F2-REQ2 |
| As a lecturer I want access to pages specific to my permissions | The side nav links will be 'Home', 'Reserve Slots', 'My Modules' and 'View Timetables' | Medium | F2-REQ3 |
| F3 | Select Modules | As a student I want to be able select my modules | Student will be able to select modules specific to their year and course. | Medium | F3-REQ1 |
| F4 | Reserve Slots | As a lecturer I want to be able to reserve slots in which I can't take classes | Lecturer clicks on time slot, and it is set to 'Reserved' | Medium | F4-REQ1 |
| F5 | View Timetables | As a user I want to be able to view my timetable | Side Nav links to page that displays users’ timetable | High | F5-REQ1 |
| As an Admin I want to be able to view all timetables. | Side Nav links to page that an Admin can select and view all timetables, for Lecturer, Student, Room, and Module | High | F5-REQ2 |
| F6 | Courses - Create, Read, Update, Delete. (CRUD) | As an admin I want to be able to Create, View, Edit and Delete Courses | Admin has access to Course page. Here they can Create, Edit, View, and Delete all courses from the table. | High | C - F6-REQ1  R - F6-REQ2  U - F6-REQ3  D - F6-REQ4 |
| F7 | Modules - CRUD | As an admin I want to be able to Create, View, Edit and Delete Modules | Admin has access to Modules page. Here they can Create, Edit, View, and Delete all modules from the table. | High | C - F7-REQ1  R - F7-REQ2  U - F7-REQ3  D - F7-REQ4 |
| F8 | Class types - CRUD | As an admin I want to be able to Create, View, Edit and Delete Class types | Within 'Modules' page, Admin can Create, Edit, View class types when doing the same for Module | High | C - F8-REQ1  R - F8-REQ2  U - F8-REQ3  D - F8-REQ4 |
| F9 | Buildings - CRUD | As an admin I want to be able to Create, View, Edit and Delete Buildings | Admin has access to Buildings page. Here they can Create, Edit, View, and Delete all buildings from the table. | High | C - F9-REQ1  R - F9-REQ2  U - F9-REQ3  D - F9-REQ4 |
| F10 | Rooms - CRUD | As an admin I want to be able to Create, View, Edit and Delete Rooms | Admin has access to Rooms page. Here they can Create, Edit, View, and Delete all rooms from the table. | High | C - F10-REQ1  R - F10-REQ2  U - F10-REQ3  D - F10-REQ4 |
| F11 | Students - CRUD | As an admin I want to be able to Create, View, Edit and Delete Students | Admin has access to Rooms page. Here they can Create, Edit, View, and Delete all students from the table. | High | C - F11-REQ1  R - F11-REQ2  U - F11-REQ3  D - F11-REQ4 |
| F12 | Lecturers - CRUD | As an admin I want to be able to Create, View, Edit and Delete Lecturers | Admin has access to Lecturers page. Here they can Create, Edit, View, and Delete all lecturers from the table. | High | C - F12-REQ1  R - F12-REQ2  U - F12-REQ3  D - F12-REQ4 |
| F13 | Schedule Course | As an admin I want to be able to create a selected course and set constraints on the schedule. | Admin has access to Schedule page. Here they can select a course and the available constraints and optimisations are displayed and able to be modified, and then schedule the course via button. | High | F13-REQ1 |
| F14 | Schedule Outcome | As an admin I want to be able to view the data from the outcome of the schedule attempt. | After schedule button is pressed from F13, status, statistics and outcomes from the schedule are shown. | High | F14-REQ1 |

Table 1 Web Applications Functional Requirements

* 1. Optimisation Engine

Software Specification

Frontend:

|  |  |
| --- | --- |
| Timetable Creation | |
| Expected Functionality | Course selected; Course ID and constraints passed to backend via JSON object |
| Unwanted Functionality | Constraints not passed |

Table 2 Software Specification - SS1

Backend:

|  |  |
| --- | --- |
| Add schedule details | |
| Expected Functionality | Schedule details row added to 'schedule' table. Status set to "pending" |
| Unwanted Functionality | Schedule not stored in schedule table - SQL Error |

Table 3 Software Specification - SS2

|  |  |
| --- | --- |
| Add constraints to database | |
| Expected Functionality | Get passed in constraints from frontend, send to 'constraints' database table |
| Unwanted Functionality | Constraints not stored in constraints table - SQL Error |

Table 4 Software Specification - SS3

|  |  |
| --- | --- |
| Schedule Controller - Get Modules | |
| Expected Functionality | Get Modules via passed in Course ID |
| Unwanted Functionality | SQL Error |

Table 5 Software Specification - SS4

|  |  |
| --- | --- |
| Schedule Controller - Get Modules | |
| Expected Functionality | Get Classes via passed in Module ID |
| Unwanted Functionality | SQL Error |

Table 6 Software Specification - SS5

|  |  |
| --- | --- |
| Set Modules as Applied in Constraint Optimisation | |
| Expected Functionality | Set Modules as Random, By Popularity, or Custom Order |
| Unwanted Functionality | Constraints not passed |

Table 7 Software Specification - SS6

|  |  |
| --- | --- |
| Set Class Order as Applied in Constraint Optimisation | |
| Expected Functionality | Classes - Lecture, Tutorial, Practical, Lab, Advisory - ordered as applied |
| Unwanted Functionality | Constraints not passed |

Table 8 Software Specification - SS7

|  |  |
| --- | --- |
| Loop through module list | |
| Expected Functionality | Starts with first module as expected through *SS6* |
| Unwanted Functionality | Module List empty |

Table 9 Software Specification - SS8

|  |  |
| --- | --- |
| Switch between classes | |
| Expected Functionality | Get class order as expected through *SS7* |
| Unwanted Functionality | Constraint not passed in |

Table 10 Software Specification - SS9

|  |  |
| --- | --- |
| If reschedule reset timetable slots | |
| Expected Functionality | If the schedule is a re-schedule rather than create schedule, reset involved parties’ slots from previous schedule. |
| Unwanted Functionality | Scheduled instead of re-scheduled, leading to slots not being replaced and all modules scheduled twice. |

Table 11 Software Specification - SS10

|  |  |
| --- | --- |
| Do schedule | |
| Expected Functionality | Schedule Engine completed as described in Section 4 |
| Unwanted Functionality | Error thrown during algorithm |

Table 12 Software Specification - SS11

*SS11* fully expanded on in section "4. Implementation."

|  |  |
| --- | --- |
| Succeeded Classes | |
| Expected Functionality | Succeeded Classes passed to 'succeeded' table in database (if applicable) |
| Unwanted Functionality | Succeeded Classes not stored - SQL error |

Table 13 Software Specification - SS12

|  |  |
| --- | --- |
| Failed Classes | |
| Expected Functionality | Failed Classes passed to 'failed' table in database (if applicable) |
| Unwanted Functionality | Failed Classes not stored - SQL error |

Table 14 Software Specification - SS13

|  |  |
| --- | --- |
| Reasons | |
| Expected Functionality | Reasons for failed classes passed to 'reason' table in database (if applicable) |
| Unwanted Functionality | Reasons not stored - SQL error |

Table 15 Software Specification - SS14

|  |  |
| --- | --- |
| Suggestions | |
| Expected Functionality | Suggestions of heuristic modifications passed to 'suggestion' table in database (if applicable) |
| Unwanted Functionality | Suggestions not stored - SQL error |

Table 16 Software Specification - SS15

|  |  |
| --- | --- |
| Schedule Details | |
| Expected Functionality | Update Schedule details row from 'schedule' table. Status set to "scheduled" or "failed" |
| Unwanted Functionality | Schedule not updated - SQL error |

Table 17 Software Specification - SS16

1. Design
   1. Architectural Description

Figure 1 is a High-Level Design of the system. There are 3 actors of Admin, Student, and Lecturer.

A screenshot of a computer

Description automatically generated

Figure 1 Skeleton System Architecture High-Level Design

As described in the Functional Requirements from *Table 1*,this design of the architecture will facilitate F3-F14,

Admins will be able to perform their scheduling duties, while also performing the CRUD commands for the data.

Students will be able to select modules and view their timetables.

Lecturers will be able to reserve slots within the week and they will also be able to view their timetables.

A backend system is needed to execute the scheduling engine.

A RESTful API is needed for the frontend to access the backend via URL routers.

This will facilitate communication from the frontend to the database, by using the routers and the CRUD SQL commands that are executed in the backend.

A relational database management system is needed to facilitate a robust data management layer, which will store the results of the schedule, and update timetables, while also storing the general data of the entities shown in Figure 1.

Figure 2 shows the initial skeleton mock up design of the database.

A diagram of a computer

Description automatically generated

Figure 2 High-Level Skeleton Database Design

* 1. User Interface Design

From Figure 1, seeing that there are many different operations that the users must carry out, a dashboard system with a side navigation bar to access these different pages would be the ideal design.

Figure 3 below shows what the side-nav will look like for the admin, and it also shows what the home dashboard page will look like, it will display various useful statistics to the user.

A close-up of a document

Description automatically generated

Figure 3 Side Nav and Home Page Mock Up

For the lecturers and students they will only have access to the pages described in Functional Requirements F2-REQ2 and F2-REQ3.

A white sheet with black text

Description automatically generated

Figure 4 View Edit and Delete Courses

Figure 4 shows what the pages will look like for Admins to View all the data. In Fig. 4 "Courses"

is the example. Note it has an "Edit" and "Delete" button beside each record that will allow those functions to be performed.

A screenshot of a computer

Description automatically generated

Figure 5 Create Course

Figure 5 shows what the "Create" pages will look like. Inputs for each attribute and a submit button. The "Edit" pages will look the same but have the fields pre-set with the current data of the chosen record.

A screenshot of a computer

Description automatically generated

Figure 6 Create Schedule Mock Up

Figure 6 is the mock-up for the "Schedule Course" page.

Create Schedule will need a dropdown of the courses for the user to select, then the course's modules will appear, and the rooms that the building the course is in.

After that, all the optimisations and modifiable constraints for the user to choose will be available to the user.

When a schedule is completed there needs to be a page that displays the outcomes of the timetable including, succeeded/failed modules, succeeded/failed classes, reasons for failed classes, and suggestions.

A screenshot of a computer

Description automatically generated

Figure 7 Timetable Outcomes Mock Up

For the simple mock up seen in Figure 6, the user selected the failed module "Environmental Biology" and the succeeded and failed classes for that module are asynchronously output.

For Advisory 2, the only available rooms left for it to be scheduled in do not have the correct facility. However, if the admin applies 75% room capacity leeway, the class will be able to be scheduled in a smaller room, in which the facility does match.

Design choices for this page have been made so that the user has an intuitive and easy to understand description of why the course failed to schedule. However, it will be less vibrant and colourful than the other pages in the application. This is because, "As a rule, interactive infographics are less vibrant and illustrated since they mostly bet on data and technical side rather than design, but this does not diminish their dignity and attractiveness". [3]

The user is given crucial insight on the schedule attempt. A pie chart is displayed for easily understandable statistics on the ratio of success to failure for both modules and their classes. They see what classes succeeded, and what slots these were assigned to, and what classes failed and why.

The user's current constraints that are applied are output and they can easily apply the suggested constraints to make the schedule work.

The requirement from SS12 is how the Succeeded Classes need to be passed to the database. The succeeded classes field in this page will get it from the database and be displayed.

This also goes for SS13. We need SS13 to succeed to display the failed classes in the timetable outcomes page.

* 1. Software System Design Component Breakdown

For the when admin Creates, Edits or Deletes an entity via the UI component it will communicate this data to the database via the backends’ RESTful API component. The RESTful API will use routing to define URL's that the frontend will call.

When the frontend calls these URL’s, it will pass the data through a JSON request object, in which the backend will handle by manipulating and managing it to execute SQL commands to the database component.

The Schedule Optimisation Engine will need to get the Lecturers, Modules, Students and Rooms, from the database. It also needs their timetables from the database so that their slots can be checked against and updated if the schedule is successful.

The Engine also needs the constraints and optimisations modified by the user in the UI from the frontend component so that these can be applied in the algorithm. A diagram of a course

Description automatically generated

Figure 8 Basic Schedule Design without Constraints/Modifications Applied

1. Implementation
   1. Choice of Implementation Language and Environment

TypeScript was the language used for both frontend and backend. This was used mainly for its compiler time error handling.

Unlike JavaScript, TypeScript introduces optional strong static typing: Once declared, a variable doesn’t change its type and can take only certain values. The compiler alerts developers to type-related mistakes, so they have no opportunity to hit the production phase. This results in less error-prone code and better performance during execution. [4]

Furthermore, it enhances code understandability because it lets the developer define the expected types of variables, function parameters, and return values. [5]

A diagram of a server

Description automatically generated

Figure 9 Development Environment and Software Libraries

**UI**

As shown in Figure 1, the User Interface will be made with Next.js. Next.js is a React framework for building full-stack web applications [6]. React is a JavaScript library that lets you build user interfaces out of individual pieces called components [7]. Therefore, Next will be used in our system to build UI and pages for the web app with React Library [8] by rendering React components and managing client-side routing.

**Backend**

The backend will consist of a Node.js runtime environment while using express.js to create the RESTful API to manage the server-side of our system.

Node uses an event-driven, non-blocking I/O model [9] this means that Node is built well to handle asynchronous JavaScript code to perform many asynchronous activities such as reading and writing to the file system, handling connections to database servers, or handling requests as a web server [10].

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for web [11] applications. For our system, express.js is used for routing. These routes are part of our Node.js environment, in which they use http requests to allow communication to the backend. They handle how an application's endpoints respond to client requests and allow our frontend server-side to do GET, POST, PUT, and DELETE calls directly from the UI's web pages by the user. Routing will take data from the frontend via 'requests' and can then 'respond.'

Express.js uses the node environment to facilitate communication from the frontend to the backend as a RESTful API.

MySQL will be used as the database management system. This is because our database - demonstrated in our basic mock-up in Figure 3 - shows how there is many relationships between the entities. Therefore, an RDBMS was needed, and MySQL was chosen due to it offering superior speed, scale, and reliability [12].

* 1. Use of Software Libraries

A screenshot of a computer

Description automatically generated

Figure 10 Next.js Node.js MySQL Interaction Overview

Figure 10 shows how the Next.js and Node.js libraries interact with each other.

To give a brief full-stack overview before delving into each separately. The frontend interacts with the backend, through the data.ts Next file. The data.ts file has a function for every call the frontend needs to make to the backend. This is done through Express. Express RESTful API is used to fetch, post, update, and delete data to/from the frontend via URL routing, being hosted on "http:localhost/8080/api/". So the data.ts file has functions such as await fetch(`http://localhost:8080/api/building/${building\_id}`) which will fetch the Building with that Building ID, because in the backend the node.js express building.routes class has defined that when that URL is called to GET the building with that building ID through operations in the building.controller and building.repository classes.

**Frontend**

A diagram of a computer program

Description automatically generated with medium confidence

Figure 11 Frontend Data Movement

In our frontend application Next.js facilitates the use of server-side rendering with its client-side rendering.

The /dashboard/ classes are server-components while the /ui/ are client-components.

Server Components allow you to move data fetching to the server, closer to your data source [13].

Client Components allow you to write interactive UI that is prerendered on the server and can use client JavaScript to run in the browser [14].

Next.js uses a filesystem-based router where **folders** are used to define routes.

Each folder represents a **route** segment that maps to a **URL** segment. To create a nested route, you can nest folders inside each other. [16]

For Figure 11, each component in the /dashboard/ section represents a directory. These directories will have a page.tsx file. These pages call the UI components as functions to display them on the web page.

So each directory corresponds to the current user URL. For example, if the admin has selected the "Buildings" page from the side nav bar, this will display the contents of /dashboard/buildings/page.tsx.A screenshot of a computer

Description automatically generated

Figure 12 Routing URL Example

Figure 12 is displaying the contents of /dashboard/buildings/page.tsx, this is a direct correlation of the URL.

The data.ts file is the direct link to the backend so it is the source of data fetching and posting to the database. With the /dashboard/ apps being server-side this is how they get data.

The server-side apps are asynchronous, and they get the backend data from data.ts.

So the /dashboard/ apps do not make direct calls to the express.js API.

Each CRUD function needed for the frontend is defined in data.ts, the /dashboard/ apps then asynchronously call whatever functions they need via await.

A diagram of a data flow

Description automatically generated with medium confidence

Figure 13 Fetch Buildings from Database

So from looking at Figure 10, this is the black line of the data.ts file grabbing data from the backend. Then looking at Figure 13, the function defined in data.ts (getBuildings()) is called by the /dashboard/buildings page (blue line connecting data to buildings), the buildings server-component page now has the list of buildings from the database.

However, this data is not being displayed to the user yet. The client components need to do this. These client components are the /ui/ files. These files are the actual web pages that the user interacts with, but again since these are client-side rather than server-side they can't make calls to the database. Therefore the data must be passed in from the dashboard files.

These UI files are functions that the dashboard files call, and they can pass in the parameters of the data it requires.

A screenshot of a computer

Description automatically generated

Figure 14 Display Buildings to Client-Side

This is the gold line from /dashboard/buildings to /ui/buildings demonstrated in Figure 11.

This is how the "View/Read" requirement REQ2 will be met from F6-F12, but we still need a way to satisfy the POST UPDATE and DELETE requirements.

This is done through the actions.ts file. While the data.ts file is used to call the express.js RESTful API URL, actions.ts is used for getting Form Data from the client-side pages and passing this client-side data to the data.ts file so that it can pass this object into the URL as a parameter or JSON body.

For example, from Figure 5 of the Create Course Mock-up, the Form Data of the inputs will be passed to actions.ts, which will be put it into the correct format of Course model, then call the createCourse() function from data.ts, which POSTs this data to the RESTful API, which then lets the backend unravel and sort out the data as necessary. To format the data in the correct way there is a file called definitions.ts which exports each entity as a "type." With matching definitions/formats defined in the frontend and backend it ensures type integrity and that the backend node.js knows exactly what the request it has been sent in the express URL is.

A diagram of a building

Description automatically generated

Figure 15 Insert Caption to Backend

Figure 10 shows how the requirements from REQ1 POST will be met for F6-F12.

The actions.ts file gets the FormData from the input fields of the client-side class, formats it as a defined object, passes it to the data.ts function, and that function calls the URL with the object as a parameter so that the backend gets it.

Looking back at our Design Mock-up for Figure 4 for view, edit, and delete design. The design choice was made to include an "Edit" and "Delete" button for every record. This choice was made as next.js has a feature called Dynamic Routing.

When we want to edit a record, we obviously won't be able to create an edit page for every single row, so we use dynamic routing. When you don't know the exact segment names ahead of time and want to create routes from dynamic data, you can use Dynamic Segments that are filled in at request time or prerendered at build time. [15]

A Dynamic Segment can be created by wrapping a folder's name in square brackets: [folderName]. For example, [id] or [slug]. [16]

When we want to edit a record, we need its ID.

For /dashboard/building/ we have created two new directories called /edit/[id]/ and a page.tsx file.

The "edit" button in our UI has an onClick() event that redirects to href:

href={`/dashboard/buildings/${building\_id}/edit`}

The selected building\_id is now in the URL.

A screenshot of a social media post

Description automatically generated

Figure 16 Dynamic Routing Edit URL

So the directory /dashboard/buildings/[id]/edit/page.tsx is dynamically now /dashboard/buildings/8/edit/page.tsx.

And through dynamic rendering the page.tsx can get this ID through params.

export default async function Page({ params }: { params: { id: string } })

Figure 16 shows this if "Edit" was pressed on building with building ID 13. (F6-F12 REQ3 will be met this way.)

A screenshot of a computer

Description automatically generated

Figure 17 Client-Side Input to Backend for Update

For deleting a record, the onClick() function of the rows corresponding delete button it can directly call an action to delete the row with that ID. The data.ts file will call the express REST API's defined delete building URL while passing in the building ID as a parameter. F6-F12 REQ4 will be met this way.

**Backend**

The backend node.js is split up into 4 directories. Routes, Controllers, Repositories, and Models.

Each table in the database that has a corresponding file in each of the directories.

The backend also includes a file for the optimisation engine; schedule.controller.ts

A diagram of a diagram

Description automatically generated with medium confidence

Figure 18 Backend Overview

The .routes files contain the express.js URL's that facilitate communication from the frontend and backend. These files define a list of http calls that the frontend can make to perform calls to the database.

For example, the building.routes.ts will list,

- POST: localhost:8080/api/building/{building}

- GET: localhost:8080/api/building/

- DELETE: localhost:8080/api/building/:course\_id

and any more routes that are required accessing and manipulating building data from the frontend.

These routes define the URL but then it calls the controller classes to handle what the actual request is from the frontend, and how to respond.

For example, when a building is created, the frontend calls the URL localhost:8080/api/building/ and passes in the building JSON object in its body.

When the frontend calls the REST API it sends a request and response, the controller classes are used to extract the JSON object from the request body (using model.ts if it’s a specific entity data type) and validate it to ensure it meets the required criteria in the correct format. If the validation is successful, it calls the corresponding method of the repository class. If this method doesn’t throw any errors the controller class sends/gets the data to/from the database.

The Repositories are where the actual SQL commands are written. It takes the body passed in by the controller and uses it to set as the values to be executed on for SQL calls. For all methods in the repository classes, if there's an error to do with the values it’s gotten from the controller it throws an error, else it resolves.

Models are similar to the definitions.ts class from the frontend. These are used with controllers and repositories to define the structure of the data. Each model represents a specific data structure or entity [17]. It provides a blueprint to the controllers and repositories on how the data should be managed, manipulated, and stored.

* 1. Key Implementation Decisions

Multiple new database tables were implemented from after the initial document:

* 'constraints' table. This is separate to the constraints that are passed in in the body. These are stored in the database after a schedule is completed so that in the outcomes page these can be viewed and amended. They are fetched from the database rather than to pass the constraints object that was passed in back from backend to the frontend again.
* Hence, 'class\_order' and 'module\_order' tables were also created with the 'constraints' table to store them, each have a Constraints ID foreign key so that they are related to constraints.
* 'pending' table implemented, stores all timetables and slots currently scheduled for a particular course, this way if course is Re-scheduled, they are reset before the new schedule.
* 'failed' stores all failed classes during a schedule attempt.
* 'reason' stores the reasons that classes failed during a schedule attempt.
* 'succeeded' stores all succeeded classes during a schedule attempt.
* 'suggestion' stores the list of modifications a user can make to soften constraints and make a schedule attempt succeed. These are displayed to the user, and they can click to apply them. User can then re-schedule with new constraint heuristics applied.
* 'schedule' stores a Schedule ID that all the aforementioned tables reference for relational purposes. It also stores the status and datetime of the schedule.
* 'modules\_to\_students' table was created. This is so that students can select a module. This table simply stores pairs of Module IDs and User IDs. When a student selects a module this specific Module ID is stored with their User ID in the same row. This was implemented so that all the students in a module could be easily attained through an SQL command using Module ID.
* 'student' and 'lecturer' tables were amalgamated into one 'user' table.

Another key implementation decision was deciding upon what constraints and optimisations to implement.

|  |  |  |
| --- | --- | --- |
| **Constraint** | **Description** | **Options/Input** |
| Traversal Order | Order in which the slots (e.g. Monday 9am to Friday 4pm) are looped through. | Start to end  End to start  Random  Day by day |
| Module Order | What module the engine schedules first to last. | By popularity  Custom - (draggable list)  Random |
| Class Order | What class types are scheduled first to last. | Draggable list of the 5 class types |
| Lecture Room Leeway | Percentage of students that need to fit in the room for a lecture. | Slider for number between 0 and 100 (%) |
| Tutorial Room Leeway | Percentage of students that need to fit in the room for a tutorial. | Slider for number between 0 and 100 (%) |
| Practical Room Leeway | Percentage of students that need to fit in the room for a practical. | Slider for number between 0 and 100 (%) |
| Lab Room Leeway | Percentage of students that need to fit in the room for a lab. | Slider for number between 0 and 100 (%) |
| Advisory Room Leeway | Percentage of students that need to fit in the room for an advisory. | Slider for number between 0 and 100 (%) |
| Lecture Student Leeway | Percentage of students that need to have the slot free for lectures. | Slider for number between 0 and 100 (%) |
| Tutorial Student Leeway | Percentage of students that need to have the slot free for tutorials. | Slider for number between 0 and 100 (%) |
| Practical Student Leeway | Percentage of students that need to have the slot free for practicals. | Slider for number between 0 and 100 (%) |
| Lab Student Leeway | Percentage of students that need to have the slot free for labs. | Slider for number between 0 and 100 (%) |
| Advisory Student Leeway | Percentage of students that need to have the slot free for advisories. | Slider for number between 0 and 100 (%) |
| Lecture Classes Per Day | Maximum number of lectures that can be scheduled in a single day. | Number Input |
| Tutorial Classes Per Day | Maximum number of tutorials that can be scheduled in a single day. | Number Input |
| Practical Classes Per Day | Maximum number of practicals that can be scheduled in a single day. | Number Input |
| Lab Classes Per Day | Maximum number of labs that can be scheduled in a single day. | Number Input |
| Advisory Classes Per Day | Maximum number of advisories that can be scheduled in a single day. | Number Input |
| Use Substitute Lecturer | Allow a modules substitute lecturer to be scheduled if main lecturer unavailable. (each module has an assigned sub selected in create) | Checkbox |

Table 18 Constraints Table

* 1. Schedule Algorithm

A screenshot of a computer program

Description automatically generated

Figure 19 Schedule Algorithm Diagram

The Schedule Algorithm lives in the schedule.controller.ts class. It is called from the frontend by the express.js POST route http://localhost:8080/api/schedule/${course\_id}.

By being a controller class it can handle the data that the frontend passed in its Request.

The frontend passes in the course ID in the URL params and it passes the constraints that have been defined in the Schedule Course web page in the req body. (Appendix A.1. for Create Schedule page)

1) **doSchedule() -** The main function is called doSchedule(). We get variable `course\_id` is got through req.params.course\_id and the object `constraints` is assigned to req.body.  
The `constraints` object is posted to the 'constraints' database, this is so that in the timetable outcomes page displayed to the user after a schedule attempt, the constraints can be displayed and heuristically modified by applying the suggestions displayed to the user.

There is a function here that if the schedule has already been scheduled (courseDetails.scheduled == true) we know that it needs to be rescheduled. Therefore, each timetable/slot pair from the 'pending' row is reset to "available" and the pending row itself is reset to null. This row contains every timetable and slot pair that was scheduled on the last schedule. They are reset so that they aren’t scheduled twice on this attempt. courseDetails.scheduled is set to false.

The modules are ordered depending on the moduleOrder constraints. The objects required for the outcome statistics are initialised as arrays: pending, failed classes, succeeded classes, reasons, and suggestions.

2) **classtype switch -** For each class type, there are separate failed reasons objects initialised. Their leeway, leeway capacity threshold and room list are initialised - separate room lists accounting for their different leeway constraints.

A for loop from 0 < 5 is started (for the 5 different class types). A switch statement is in this for loop, so that they are scheduled in the order of the class order constraint.

Each switch starts the scheduleClass() method. The different room lists, failed reasons, and number of classes for the separate classes are passed into this, as well as the essential standard data like course details, module details, lecturer, and outcome-required objects.

3) **scheduleClass() -** Variables are initialised based on the course, module (module name, module ID), and constraints parameters passed in. Variable `count` is assigned to the number of classes. If count is 0, false is returned. So if a module has no classes for a class type, scheduleClass is immediately broken out of and the next switch case is executed ( see '2)' ). Variables required for '5)' are set. See appendix A.2.

4) **slots array -** `slots` array string variable is set depending on traversal order constraint. Note that the index for this loop is called `tts.` Here again if count == 0, loop is broken out of (all classes for this class have been scheduled).

5) **classes per day logic** - If the traversal constraint is not `random` or `day by day`, it means the user has customised the maximum number of classes per day constraint. The `currentHourIndex` is set to the index of the current slot (`tts`) mod `numHoursPerDay` (8 - indicating hours between 9:00 to 17:00).

If `skipToNextDay` is true it means we want to skip to the next day (see '10)' ). We do this by calculating the `skipCount` by doing `numHoursPerDay` - `skipCount` this calculates the number of times the array has to be looped through to get to a new day (M9,T9,W9,Th9.F9). `skipToNextDay` is set to false.

If `skipCount` is > 0 it means that there are more hours left in the day. We don't want to process these slots, the slots loop is continued and it gets to this logic again, and `currentHourIndex` is calculated again.

However, if it is the last slot of the day (`tts` == 39 && count > 0) we break out of the main loop if we are already on are `secondPass`. Else, we start the loop over again. We set `secondPass` to true and tts to 0 and go back to the beginning of the loop. If `secondPass` is false it means that the number of classes per day was not able to be adhered to, so the loop must be traversed again. We indicate this by setting `secondPass` to true so that this action is not executed again. `tts` is continued.

If the updated `currentHourIndex` is 0, it means we are at the start of a new day, so `classesScheduledToday` is set to 0 (see A.3 and '10)').

6) **lecturer availability** - The lecturerAvail() class is called. It will carry out the lecturer availability logic and return true or false. If it returns false `slots` is continued. This function simply checks the lecturer’s timetable for that slot to see if it does not equal "available". If it doesn't equal "available" it means the lecturer already has a class scheduled or has reserved it at that slot. The lecturers `user\_id` `firstname` and `lastname` are taken and string variable `recentFailure` is explains the reason for failure.

The `reason` object is set (based on reason.model.ts) and it is assigned to an object containing the `schedule\_id`, `recentFailure`, `module\_id` and `classtype` and is passed to handleFailedReason(). handleFailedReason() is a function that checks if this reason is already a part of the reasonArray (declared in '1)' ) and if so, it is added to the array, else it takes `slot` from the passed in reason object and appends it to the pre-existing failedReason in the array. This prevents huge amounts of duplicate reasons, which only differ due to `slot.` There will only be one record for each reason of each class type (and class number)

7) **find room logic** - If a slot for lecturer has successfully been found we now check for a room. The `roomList` acquired in '1)' is all the rooms that fit the capacity leeway threshold (Appendix A.4). Variable `foundRoomIndex` is assigned to the outcome of the function roomAvailIndex(). roomAvailIndex() loops through the `roomList` with index `r`. It checks room[r]'s timetable to check if it is free at the slot we are currently at within the outer loops scope. If unavailable it calls handleFailedReason(), sending in the same data as before in '6)' but now the reason is different, it is now room occupied rather than lecturer. If a room has been found at the slot it now checks to see if the class facility matches the room facility. Again, if not, handleFailedReason() is called with the reason being how the room does not match, specifying class facility and room facility in string. If a room facility does match the 'r' index is returned. foundRoomIndex = r.

8) **unavailable student logic** - The `unavailableStudentThreshold` is calculated (Appendix A.5). From this, the maximum number of unavailable students is calculated. Then, `unavailableStudentCount` is assigned to the return result of function studentUnavailCount(), in which the list of studentTimetables and the current slot is passed in. This function simply; sets an `unavailableStudentCount` to 0, loops through each student’s timetable, and if the slot in the timetable is not equal to "available" `unavailableStudentCount` is incremented by 1. It is then returned after every student's timetable in this module for this slot has been looped over. If the unavailableStudentCount is greater than maxUnavailStudents the loop is continued to the next slot.

9) **found slot procedure** - If the program has got to this point the lecturer's, room's, and module's timetables are updated for the current slot with the module name and classtype in the database. These timetable\_id and slot pairs are added to the `pendingList.` This means that if any module fails at any point, this `pendingList` can be used to grab each scheduled timetable and slot and reset them back to "available." All the students in this module are looped over and the same process is executed for all of them. `foundSlot` is set to true.

10) **end of original loop** **(success)** - This class has successfully been scheduled. The details of this class are formatted into model 'succeededClasses' with `schedule\_id`, `user\_id` (lecturer), `module\_id`, `room\_id`, `classtype`, and `slot.` It is then pushed to the `succeededClasses` array.

At this point classes per day logic is executed. `classesScheduledToday` is incremented, and if ` classesScheduledToday ` is equal to `classesPerDay`, 'skipToNextDay' is set equal to true. This means when we get to classes per day logic ( '5)' and A.3) the logic is carried out so that the index of the slots array is set to the corresponding index of M9 T9 W9 Th9 or F9.

11) **room slot suggestion handling (indicates failure) -** At this point we've reached the end of the loop. If `foundSlot` == false it means the entire loop has been traversed and the class was unable to be scheduled. The `failedReasons` array (populated in the handleFailedReasons method calls) is looped over and posted to the database. Boolean variable `facilityRoomAvailable` is set to false. `roomList` is updated, it now retrieves every single room in the building that has the correct facility no matter capacity. This is done so that the necessary leeway required to make this class fit in this room is calculated, another slots loop is created and for every slot facRoomAvailIndexSuggestion() method is ran. This loops through each room in the new roomList, and if the room is available, it executes the handleSuggestion() method, which adds to the `suggestions` array the required leeway to make the students fit in the room so that it can be scheduled. If there is a room available `facilityRoomAvailable` is set to true.

12) **student leeway logic and suggestion handling -** If `facRoomAvail` is false, it means we must soften the student availability constraint. This is done by modifying the studentLeeway for this particular class. Another loop of the slots is executed. Within this, it calculates the `unavailableStudentCount` from the same method as in '8)' and through this unavailableStudentCount it calculates the required leeway for this slot to be scheduled (A.6). Again, this means the percentage of students that must have that slot free, so if this constraint is softened a percentage of students will have conflict/overlap of classes at this slot but it will be scheduled anyway. If the `unavailableStudentCount` is greater than `maxUnavailStudents` for the current slot handleFailedReason() is executed, with the reason being about how many students are unavailable and also what the current leeway is. handleSuggestion() is then executed in regard to the required leeway that is needed to schedule the slot. This is implemented at the bottom of scheduleClass() as it should only be done as a last resort.

13) **failed class handling and scheduleClass returned** - `failedClassNum` is calculated as the number of classes for this class minus `count.` If the current class we are on has failed, we know every other class of this specific type for this module will fail. By calculating `failedClassNum` we loop through the number of classes left for this class and add them to `failedClasses` array with the number of the class.

The value of `scheduleFailed` is returned, either true or false.

The result of scheduleClass() is stored as `thisSchedFailed`. If `thisSchedFailed` is true, we set variable `scheduleFailed` in the doSchedule() scope to true. This way the program knows that the schedule has not succeeded as at some point in the program a class failed to schedule. scheduleClass() is still ran for the rest of the modules to gain all the failed and succeeded classes, with reasons and suggestions.

14) **use substitute lecturer logic** - If schedule failed has failed and constraint `useSubLecturer` is true, the `failedClasses` are looped over executing the scheduleClass() method this time with each modules substitute lecturer passed in as the lecturer. As stated, the failedClasses are looped over rather than modules. The switch statement is still used but not looped over, each count is set to 1, the switch statement happens for every failed class. Again, this carries out the exact same scheduleClass() method, but with the sub lecturer. Indicated by this part being at the end of the engine; this is considered a last resort.

15) **schedule success** - With the course successfully scheduled, the course is updated in the database to have `scheduled` = true. The schedule row in the database also is updated to have status as "scheduled." The `pendingList` is joined as a string with a key separating, each `timetable\_id|slot` pair and posted to the database. In the UI the Timetable Outcome page is displayed, displaying statistics on the schedule and the timetable for each module, room, student, and lecturer.

16) **schedule failed -** succeeded classes posted to the database, failed classes posted to the database, suggestions posted to the database (use substitute lecturer logic done here, but in this case the useSubLecturer constraint is false). The pendingList is looped over and resets each scheduled timetable and slot over the course of this algorithm back to "available", the row in the database is set to null. Course is updated in the database to have scheduled = false. The schedule row in the database also is updated to have status as "failed."

In the UI the Timetable Outcome page is displayed to the user, displaying statistics, succeeded modules, failed modules, succeeded classes, failed classes, and suggestions. These suggestions can be applied by the user, and they then can choose to re-schedule with these new heuristics applied. Users can click on a succeeded or failed module and show the succeeded and/or failed classes from it. They can see where each class that succeeded was scheduled. This is shown below in Figure 20.

A screenshot of a computer

Description automatically generated

Figure 20 Timetable Outcome Page

1. Testing

The testing of the application involved a combination of system testing and integration testing.



Postman was used throughout the development of the system to carry out continuous and rigorous testing of the API endpoints.

Integration testing was used to integrate the different constraints and optimisations together overtime to ensure they work together.

For the UI all form entry input fields are validated before submitted, the form will not be submitted if the inputs are empty or in the wrong format (A.10).

* 1. Postman System Testing

Postman enables you to easily explore, debug, and test your APIs while also enabling you to define complex API requests for HTTP, and REST. [18]

A screenshot of a computer

Description automatically generated

Figure 21 Postman POST example

Figure 21 is an example of module and classtype creation, module and classtype creation are done together on the same page in the UI. Postman allows you to test this by adding data to the raw JSON body and send the POST to test and review how the backend reacts to this data. Postman will output in its response terminal the defined error that was set in the backend so that you know what to test to fix it, whether incorrect SQL syntax, connection errors, or more.

By testing the functionality and correctness of every API endpoint with Postman throughout the entire project (A.7), the backend system testing was fully covered.

Figure 22 shows the test coverage of all the endpoints calls by month.

Figure 22 Backend API Endpoint Coverage by Month

Table 18 is the total number of requests per month.

|  |  |  |  |
| --- | --- | --- | --- |
| **December** | **January** | **February** | **March** |
| 134 | 226 | 107 | 140 |

Table 19 Total Postman requests per month

Table 19 shows a heat map of the frequency of calls to the different endpoint at dates over the course of testing. This also provides an insight into how over the course of development, design changes were implemented to alter the database. For example, how the different Schedule Tables were added to allow for tracking of timetables outcomes, and how 'Student' and 'Lecturer' were merged into 'User.'



Table 20 Backend API Call Test Coverage by Day

Figure 23 shows the coverage of each CRUD request in Postman.

Figure 23 Postman coverage of different CRUD types

This methodology of continuous Postman system testing passes requirements F6-F13 REQ1-4.

* 1. System Testing

System testing involves evaluating how the various components of an application interact in a fully integrated system. It is carried out on the entire system under either functional or design requirements. [19] The specifications and requirements for this system can be found in section 2.1, these tests are based off these requirements. Two passes of system tests were carried out and by the second all tests were passing.

**First pass**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Test Description** | **Expected Result** | **Actual Result** | **Requirement Reference** |
| **1** | Go to the website url | "Login" and "Home" buttons appear | Works as expected |  |
| **2** | Select "Admin" then "Home" button | Redirected to dashboard | Works as expected | F1-REQ1 |
| **3** | Select "Lecturer" or "Student" button | Users appear, select user, select "Home" redirected to dashboard | Works as expected | F1-REQ2  F1-REQ3 |

Table 21 Home Page Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Test Description** | **Expected Result** | **Actual Result** | **Requirement Reference** |
| **1** | Correct nav-links are shown and accessible. | "Select Modules" "View Timetable" are shown. | Works as expected | F2-REQ2 |
| **2** | Select Modules from course year | Modules selected and submitted. Can't reselect submitted modules. | Works as expected | F3-REQ1 |
| **4** | Click on "View Timetable" link | Directed to Timetable page, timetable shown | Works as Expected | F5-REQ1 |

Table 22 Student Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Test Description** | **Expected Result** | **Actual Result** | **Requirement Reference** |
| **1** | Correct nav-links are shown and accessible. | "Reserve Slots" "View Timetable" and "Modules" are shown. | Works as expected | F2-REQ3 |
| **2** | Click on "Reserve Slots" nav-link and reserve slots | Slot selected and slot is displayed as "Reserved" and coloured red | Works as expected | F4-REQ1 |
| **3** | Click on "Modules" nav-link | Directed to Modules page, Lecturer's modules shown. | Works as Expected |  |
| **4** | Click on "View Timetable" link | Directed to Timetable page, timetable shown | Works as Expected | F5-REQ1 |

Table 23 Lecturer Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Test Description** | **Expected Result** | **Actual Result** | **Requirement Reference** |
| **1** | Correct nav-links are shown and accessible. | "Schedule" "Timetables" "Courses" "Modules" "Buildings" "Rooms" "Students" "Lecturers" shown | Works as expected | F2-REQ3 |
| **2** | Create / Edit Course | Fields validated; data sent. | Works as expected | F6-REQ1  F6-REQ3 |
| **3** | Create / Edit Module and Classtype | Fields validated; data sent. | Works as expected | F7-REQ1  F7-REQ3  F8-REQ1  F8-REQ3 |
| **4** | Create / Edit Building | Fields validated; data sent. | Works as expected | F9-REQ1  F9-REQ3 |
| **5** | Create / Edit Room | Fields validated; data sent. | Works as expected | F10-REQ1  F10-REQ3 |
| **6** | Create / Edit Students | Fields validated; data sent. | Works as expected | F11-REQ1  F11-REQ3 |
| **7** | Create / Edit Lecturers | Fields validated; data sent. | Works as expected | F12-REQ1  F12-REQ3 |
| **8** | Delete Course | Course deleted. | Works as expected | F6-REQ4 |
| **9** | Delete Module | Module and its related classtype deleted. | Works as expected | F7-REQ4  F8-REQ4 |
| **10** | Delete Building | Building deleted. | Works as expected | F9-REQ4 |
| **11** | Delete Room | Room deleted. | Works as expected | F10-REQ4 |
| **12** | Delete Student | Students deleted. | Works as expected | F11-REQ4 |
| **13** | Delete Lecturer | Lecturer deleted. | Works as expected | F12-REQ4 |

Table 24 Admin Home and CRUD Functions Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Test Description** | **Expected Result** | **Actual Result** | **Requirement Reference** |
| **1** | Select Course | Course's modules and rooms displayed | Works as expected |  |
| **2** | Apply a traversal order option | Timeslots traversed through in user specified choice. | Works as expected |  |
| **3** | Apply a module order option | Modules scheduled in user specified choice. | Works as expected | SS6 |
| 4 | Apply a class order constraint | Classes scheduled in user specified choice. | Works as expected | SS7 |
| **5** | Apply a room leeway | Can schedule smaller rooms dependent on leeway value | Works as expected |  |
| **6** | Apply a student availability leeway | Can schedule conflicted student slots dependent on leeway value | Works as expected |  |
| **7** | Apply use substitute lecturer | Substitute lecture is scheduled if needed. | Works as expected |  |
| **8** | On timetable outcome page | Succeeded and failed modules displayed, user can click on module to see its failed classes if any classes were successful, where they were scheduled. | Works as expected | SS12  SS13  SS14  SS15 |
| **9** | Apply suggestion | Suggestions are displayed and user presses the Apply button to apply. Constraint value changed in outcomes page. | Works as expected |  |
| **10** | Reschedule lecture on success. | Button text is "reschedule." Scheduled timetable slots reset, and course is scheduled again. | Timetables were not reset. Previously scheduled slots remained and were still considered scheduled on reschedule by engine. | SS10 |
| **11** | Reschedule lecture on failure from outcomes page. | Button text is "reschedule." Scheduled with new heuristics if they were applied. | Works as expected |  |

Table 25 Schedule Testing

**Second Pass**

After debugging the schedule engine with logging, it was made clear that the error was due to comparing if (course.scheduled === true) { // reset timetables }. Even when course scheduled was true, this if block was not being entered. This is because the MySQL database interprets 'true' as '1', and even though these both represent truthiness, the TypeScript comparator '===' compares both the values and the data type. Therefore, '==' needed to be used instead of '===' due to TypeScript’s strict data type handling. Despite, the fact both 'true' and '1' represent a Boolean truth; with '===' they needed to be the exact same value and data type. (A.9).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Test Description** | **Expected Result** | **Test Run 1 Result** | **Test Run 2 Result** |
| **11** | Reschedule lecture on success. | Button text is "reschedule." Scheduled timetable slots reset, and course is scheduled again. | Timetables were not reset. Previously scheduled slots remained and were still considered scheduled on reschedule by engine. | Works as expected |

1. System Evaluation and Experimental Results

As identified in the introduction and throughout the write-up, the aim of this project was to create a system that allowed for classes to be scheduled in the university setting. For this to happen an administrative role had to be created to facilitate the management of entities in a university, while the student and lecturer entities were facilitated to carry out personal management and viewing of their data. The scheduling of courses was the cornerstone of the system, serving as its primary objective and focal point with a smart algorithm needed to be created to allow the user to apply optimisations and constraints to a schedule.

The solution provided an intuitive and user-friendly UI in which the various sections of the web page were easy to understand and navigate. Administrators could carry out their CRUD roles seamlessly with no errors, the actions performed immediately updated the tables that the user sees.

The scheduling engine was completely successful in taking the different optimisation values and using them correctly in the algorithm.

However, there were some conclusive faults in the system. When considering the scheduling engine, it definitely could have been more efficient. Within the main loop of the slots, there could potentially be two more loops of the slots at the same incrementation, a loop for gaining suggestions of rooms, and a loop for getting suggestions on student leeway. With a huge dataset like most universities have, this will be an obvious downside to the system, it will be noticeably slow. Ideally, you would want to loop the suggestions starting at the current slot of the outer loop, but in order to get full details of the suggestions, the entire slot loop needs to be looped over again, even though we already know what the outcome is up to the point of the current slot.

When comparing this system to similar ones, the web application has a clear downside; there is no login system involved. It is fair to say that basically all universities require login to view your timetable, a quick google search of "Timetable Student Login" immediately displays university sites requesting login. [20] [21] [22]

From the Initial Project Description, an Agile methodology was used. Through the defined sprints incremental modifications were made to the prototype and this facilitated a flexible and adaptable approach to development. Postman played a crucial role in this through its quick and easy continual testing. The Agile methodology allowed for design decisions to be made but did not hinder the development of the system. For example, during a review of an upcoming sprint, the decision was made not to implement a login screen. This decision was essential in ensuring that the scheduling engine was fully functional and able to handle all constraints.

Societally, I believe it is essential that automated scheduling systems can take as much modifications and constraints as they are able to, to consider human well-being when scheduling. I think that systems like these can be incredibly beneficial to students and lecturers’ life. They can change the way an entire academic year of life goes for a person by being able to modify the weekly timetable to ensure a less stressful time, an optimised teaching experience and facilitate a great work-life balance for both students and lecturers.

1. Appendices

**User Manual**

*For both frontend and backend Run 'npm install' to install the dependencies*

* Clone the schedulerbackend Repository from GitLab at https://gitlab2.eeecs.qub.ac.uk/40293936/schedulerbackend
* Run schema files from repository
* Edit *src/config/db.config.ts* if needed and also *src/index.ts* if you wish to change the host
* Open a terminal at the root /schedulerfrontend/ directory
* Run 'npm run start'
* Clone the schedulerfrontend Repository from GitLab at https://gitlab2.eeecs.qub.ac.uk/40293936/schedulerfrontend
* Open a terminal at the root /schedulerfrontend/ directory
* Run 'npm run dev'
* App running at localhost:/8080/

**Appendices**

A.8

A screenshot of a computer

Description automatically generated

A.2

A screenshot of a computer

Description automatically generated

A.7 Postman First History December

A screenshot of a computer

Description automatically generated

# Bibliography

|  |  |
| --- | --- |
| [1] | T. U. o. Queensland, “Faculty of Health and Behavioural Sciences,” 9 January 2023. [Online]. Available: https://habs.uq.edu.au/blog/2022/10/how-avoid-and-deal-university-burnout. [Accessed 5 April 2024]. |
| [2] | A. P. Association, “APA Dictionary of Psychology,” 19 April 2018. [Online]. Available: https://dictionary.apa.org/burnout. |
| [3] | N. Birch, “Infographics and Statistics in Website Design,” designmodo, 19 May 2013. [Online]. Available: https://designmodo.com/infographics-statistics-web-design/. [Accessed 7 April 2024]. |
| [4] | altexsoft, “The Good and the Bad of TypeScript,” 14 February 2020. [Online]. Available: https://www.altexsoft.com/blog/typescript-pros-and-cons/. [Accessed 8 April 2024]. |
| [5] | M. Majdak, “What Does a Test Written with Test-Driven Development Represent: Unpacking the Benefits and Pitfalls of TypeScript,” 18 July 2023. [Online]. Available: https://startup-house.com/blog/benefit-of-typescript. [Accessed 8 April 2024]. |
| [6] | T. U. O. Q. Australia, “Faculty of Health and Behavioural Sciences,” 9 Januaray 2023. [Online]. Available: https://habs.uq.edu.au/blog/2022/10/how-avoid-and-deal-university-burnout. [Accessed 5 April 2024]. |
| [7] | Vercel, “Next.js,” 2024. [Online]. Available: https://nextjs.org/docs. [Accessed 6 April 2024]. |
| [8] | React, “react.dev,” 2024. [Online]. Available: https://react.dev/. [Accessed 6 April 2024]. |
| [9] | J. Patadiya, “Next JS vs React : Which Framework to choose for Front end in 2024?,” Radix, 1 April 2024. [Online]. Available: https://radixweb.com/blog/nextjs-vs-react. [Accessed 6 April 2024]. |
| [10] | Node.js, “About Node.js,” 2024. [Online]. Available: https://nodejs.org/en/about. [Accessed 6 April 2024]. |
| [11] | C. Team, “What is Node?,” codeacademy, 2024. [Online]. Available: https://www.codecademy.com/article/what-is-node. [Accessed 6 April 2024]. |
| [12] | Express, “Express,” 2017. [Online]. Available: https://expressjs.com/. [Accessed 6 April 2024]. |
| [13] | Next.js, “Server Components,” https://nextjs.org/docs/app/building-your-application/rendering/server-components, 2024. [Online]. [Accessed 7 April 2024]. |
| [14] | Next.js, “Client Components,” Next.js, 2024. [Online]. Available: https://nextjs.org/docs/app/building-your-application/rendering/client-components. [Accessed 7 April 2024]. |
| [15] | Next.js, “Dynamic Routes,” 2024. [Online]. Available: https://nextjs.org/docs/app/building-your-application/routing/dynamic-routes. [Accessed 7 April 2024]. |
| [16] | Next.js, “Defining Routes,” 2024. [Online]. Available: https://nextjs.org/docs/app/building-your-application/routing/defining-routes. [Accessed 7 April 2024]. |
| [17] | A. &. N. js, “Model class in Nodejs,” 27 August 2023. [Online]. Available: https://medium.com/@developerom/model-class-in-nodejs-451092ca0c93#:~:text=1-,Creating%20a%20model%20class%20in%20Node.,CRUD)%20instances%20of%20that%20entity.. [Accessed 7 April 2024]. |