

# Visualising Attendance Monitoring Data

An investigation and prototype

**Daniel Bates**

Candidate 234558

Supervised by Dr Paul Newbury

Final Report  
Computer Science MComp



School of Engineering and Informatics  
University of Sussex  
2023

### **Statement of Originality**

This report is submitted as part requirement for the degree of Computer Science Integrated Masters (MCOMP) at the University of Sussex. It is the product of my own labour except where indicated in the text. This report may be freely copied and distributed provided the source is acknowledged. I hereby give permission for as copy of this report to be loaned out to students in future years.

A handwritten signature in black ink, appearing to read 'Daniel Bates', with a long horizontal flourish extending from the end.

Signed:

Daniel Bates

### **Acknowledgements**

Thank you to my project supervisor, Dr Paul Newbury for his invaluable guidance, to the authors of the open source projects that is project is built upon, and to fellow students at the University of Sussex that provided feedback to help evaluate the project.

## Abstract

This project investigates a prototype solution to the growing problem of tracking university attendance. This is motivated by the link between attendance and attainment in higher education, since keeping track of students who may be falling behind and addressing the reasons for this, is likely to yield greater attainment and improve student experience. There is currently no widely available open-source attendance-tracker, nor an option widely in use for the University of Sussex, as most solutions are bespoke tailored options hidden behind paywalls and consulting. This project seeks to produce a three-tier application to store attendance monitoring data for visualisation to key university attendance monitoring staff. The projects objectives were generally achieved, but there is still further work to be done in this area.

Functionality produced during this project includes a Flask API with a variety of endpoints for retrieving attendance monitoring data from a PostgreSQL Database, and a ReactJS Front-end that presents several visualisations and options to filter this data.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Problem Area . . . . .	5
1.2	Objectives . . . . .	6
1.2.1	Primary Objectives . . . . .	6
1.2.2	Extensions . . . . .	6
1.3	Motivations . . . . .	6
1.4	Overview . . . . .	7
<b>2</b>	<b>Professional and Ethical Considerations</b>	<b>8</b>
2.1	BCS Code of Conduct . . . . .	8
2.2	User Testing . . . . .	9
<b>3</b>	<b>Requirements Analysis</b>	<b>11</b>
3.1	Background Research . . . . .	11
3.1.1	Similar Systems . . . . .	11
3.2	Chosen Technologies . . . . .	13
<b>4</b>	<b>Requirements Specification</b>	<b>14</b>
4.1	Mandatory Requirements . . . . .	14
4.2	Desirable Requirements . . . . .	15
<b>5</b>	<b>Design</b>	<b>16</b>
5.1	Database Design . . . . .	16
5.1.1	'Student' Table . . . . .	16
5.1.2	'Snapshot' Table . . . . .	17
5.1.3	'Course' Table . . . . .	18
5.1.4	Table Relationships . . . . .	18
5.2	Back-end Design . . . . .	18
5.3	Front-end Design . . . . .	19
5.3.1	Visualisations . . . . .	19
5.3.2	User Interface . . . . .	21
<b>6</b>	<b>Implementation</b>	<b>24</b>
6.1	Version Control Issue . . . . .	24
6.2	Debugging between three Subsystems . . . . .	24
6.3	PostgreSQL Database . . . . .	24
6.3.1	Setup . . . . .	24
6.3.2	Integration with Back-end . . . . .	25
6.4	Flask Back-end and API . . . . .	26
6.4.1	Importing Data . . . . .	26
6.4.2	API Endpoints . . . . .	26
6.4.3	Caching . . . . .	28
6.4.4	Integrating a Flask / React Project . . . . .	29
6.5	React Front-end . . . . .	30
6.5.1	Homepage/Dashboard . . . . .	31
6.5.2	Filtering Page . . . . .	31
6.5.3	Visualisations . . . . .	31
<b>7</b>	<b>Evaluation</b>	<b>34</b>

7.1	Testing . . . . .	34
7.2	Usability Study . . . . .	34
7.2.1	Methods . . . . .	34
7.2.2	Results and Evaluation . . . . .	36
7.3	Evaluation Overview . . . . .	38
7.3.1	Mandatory Requirements . . . . .	38
7.3.2	Desirable Requirements (Extensions) . . . . .	39
<b>8</b>	<b>Conclusion</b>	<b>40</b>
8.1	Objectives . . . . .	40
8.2	Retrospectives . . . . .	40
8.3	Further Work . . . . .	40
<b>A</b>	<b>Raw Usability Study Response Data</b>	<b>45</b>
<b>B</b>	<b>Project GitHub Repository</b>	<b>45</b>
<b>C</b>	<b>User Testing Compliance Form</b>	<b>45</b>
<b>D</b>	<b>Project Proposal</b>	<b>51</b>

# 1 Introduction

## 1.1 Problem Area

Attendance and attainment in higher education has a proven link, and several studies have found a correlation between the two[1][2][3][4]. In the below figure, though the axis are somewhat unclear, a clear correlation between attendance and attainment can be seen.

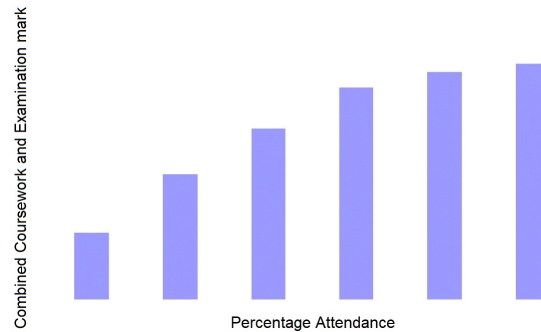


Figure 1.1: Class of combined coursework and examination mark versus percentage attendance at teaching sessions for those submitting work. (n=153); From a Computing module at the University of Central England, Birmingham[1].

There is a large task found by school administrators in monitoring attendance in order to provide support for students, as this is often for whole schools. As of 2020/21, the University of Sussex had 18,510 full-time students and 10 Schools[5]. With an approximate 18,510 full-time students, and assuming one staff member monitoring attendance per school, we can get a rough idea of the number of students monitored by an individual:

$$\frac{18510}{10} = 1851$$

This disparity is only going to increase: nationwide university uptake is only increasing in this and recent years[6], and Sussex experiences this as much as other universities[5].

This also ignores that in monitoring attendance, staff want to encourage students to improve it during the semester, not after. So regular updates or snapshots will be wanted in order to check in at several points in the semester.

Therefore, with multiple checks per semester, almost 2000 students each, and with most of these staff taking on other roles like Student Experience, this problem area is ripe for automation and likely in need of useful visualisation for the sake of users (relevant staff), as well as to enable users to identify and support students who are not engaging, and therefore improve student welfare.

## 1.2 Objectives

This project aims to produce meaningful and actionable visualisations from sample data (though if put into production, will need to be suited for live data), as well as allowing for easily configurable filters on this data that allow staff to produce their own such visualisations.

### 1.2.1 Primary Objectives

1. Import attendance data into an effective and efficient database
2. Construct a well-structured and efficient database to store attendance data, and import sample attendance data
3. Create meaningful visualisations of attendance statistics for use by relevant staff
  - (a) Per Student
  - (b) Per Degree
  - (c) Per Stage
  - (d) Per Department & School
4. Get user feedback from relevant staff: in this case the Director of Student Experience, Dr Kate Howland, who due to her background in Interaction Design and as a user may be an asset in terms of specific feedback.
5. Apply this user feedback to improve visualisations and the system at large
6. Allow users to apply filters to data and create visualisations with this dataset

### 1.2.2 Extensions

1. Create a full-stack web application around the visualisations, featuring
  - (a) A ReactJS front-end
  - (b) A Python (Flask) back-end
  - (c) PostgreSQL Database
2. Make the application user configurable, including different dashboards of visualisations
3. Investigate further ways to visualise attendance data e.g. Nightingale Rose Charts
4. Find meaningful statistics for student engagement beyond just quantity

## 1.3 Motivations

This project sets out to research and deliver useful attendance visualisations to aid staff (primarily at the University of Sussex, but this could be extended beyond) concerned with attendance.

As it is based around a web application, database, and data analysis, this project

closely relates to my degree (Computer Science Integrated Masters) and has potential to inform decisions for my masters year, as well as my preferred career in Software Engineering, as that much like the project, would include plenty of project and time management opportunities, as well as design around users which is currently being informed by the Human-Computer Interaction module. This project will allow me to test and improve my skills in Databases following the module in second year, revise and extend my statistics knowledge from Mathematical Concepts, and investigate the domain of Data Science. In addition, pursuing this project has potential to benefit the School of Engineering and Informatics if visualisations and the final application prove genuinely useful in the long term for attendance monitoring.

## **1.4 Overview**

The rest of this report details the further planning and execution of this project:

Chapter 2 covers the professional considerations.

Chapter 3 analyses requirements and conducts background research.

Chapter 4 refines the project objectives and analysis in to a requirements specification.

Chapter 5 details the project design.

Chapter 6 describes the implementation of features based on the specified requirements.

Chapter 7 evaluates the project with testing and a usability study.

Chapter 8 concludes and reflects on the project as a whole.



## 2 Professional and Ethical Considerations

### 2.1 BCS Code of Conduct

Sections of the BCS Code of Conduct[7] relevant to this project are:

**1.a. have due regard for public health, privacy, security and wellbeing of others and the environment.**

All data used in this project will be anonymised and is already held by the University in accordance with data protection legislation. This project may in future be used to identify real at-risk students (as allowed by the University through agreements made with students during admission), however no current non-anonymised students will be identified in the course of this project. The project will include user testing by a relevant member of staff, as well as the project supervisor, see 2.2.

**1.b. have due regard for the legitimate rights of Third Parties.**

This project does not engage third parties, but will make significant effort to avoid misuse of student data albeit anonymised. Any and all third-party code and libraries will be properly acknowledged.

**1.c conduct your professional activities without discrimination on the grounds of sex, sexual orientation, marital status, nationality, colour, race, ethnic origin, religion, age or disability, or of any other condition or requirement.**

No protected characteristics will be exposed in the sample data, as it has been anonymised. Therefore this project cannot and will not discriminate on any of these grounds.

**1.d promote equal access to the benefits of IT and seek to promote the inclusion of all sectors in society wherever opportunities arise.**

Effort will be made to accommodate screen-reading technologies in HTML to make the application accessible to those affected by blindness, however this will be limited to text such as tables as visualisations like graphs are unlikely to translate well. There is no auditory element planned for the application so it is unlikely to need adaptation for deaf users, and if the user interface is intuitively made, unlikely to need adaptations for users with motor disabilities either.

**2.d ensure that you have the knowledge and understanding of Legislation and that you comply with such Legislation, in carrying out your professional responsibilities.**

I understand data protection regulations, and will not make any decisions that may impose upon an individuals right to data protection and security without due consultation with the project supervisor and any other relevant persons.

**2.e. respect and value alternative viewpoints and, seek, accept and offer honest criticisms of work.**

User testing and supervisor feedback will be presented in full, and any alternative viewpoints will be considered.

**2.g. reject and will not make any offer of bribery or unethical inducement.**

There will be no incentives and no knowledge withheld from users during user feedback. See 2.2 for more.

**3.d NOT disclose or authorise to be disclosed, or use for personal gain, or to benefit a third party, confidential information except with the permission of your Relevant Authority, or as required by Legislation.**

All information given to me for use in this project will not be shared under any circumstances, except for supervision.

## **2.2 User Testing**

It has been determined that this project would comply with all points on the User Testing Compliance Form[8], and the completed form can be found in the User Testing Compliance Form C. The following section acknowledges the 12 points of the User Testing Compliance Form and gives reason why this project complies with each:

**1. Participants were not exposed to any risks greater than those encountered in their normal working life.**

Due to the nature of the project, what is shown to participants is certain to be a fairly simple interface on a web-page, which is not a risk greater than most people's normal working lives.

**2. The study materials were paper-based, or comprised software running on standard hardware.**

Materials will be software running on standard hardware (likely a networked computer in Chichester I, or a personal laptop).

**3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project.**

I will require participants to sign an informative consent form before taking part, notifying them that their feedback will only be used to improve the project, and will not be shared or published.

**4. No incentives were offered to the participants.**

Participants will not be offered any incentives.

**5. No information about the evaluation or materials was intentionally withheld from the participants.**

The introductory script will inform the participants everything relevant to the evaluation of the project.

**6. No participant was under the age of 18.**

I will not engage participants under the age of 18.

**7. No participant had a disability or impairment that may have limited their understanding or communication or capacity to consent.**

I will not engage participants with a limited understanding or capability to consent to participating in user testing.

**8. Neither I nor the project supervisor are in a position of authority or influence over any of the participants.**

Both I and my supervisor will only seek out participants we are on an equal hierarchical footing with (Student-Student, Staff-Staff only), to avoid having any authority over participants.

**9. All participants were informed that they could withdraw at any time.**

All participants will be informed of this in the introduction script, and consent form.

**10. All participants have been informed of my contact details, and the contact details of my supervisor.**

All participants will be informed of this in the introduction and debriefing scripts, consent form, as well as in any introductory email.

**11. The evaluation was described in detail with all of the participants at the beginning of the session, and participants were fully debriefed at the end of the session. All participants were given the opportunity to ask questions at both the beginning and end of the session.**

The evaluation will be described in detail in the introduction and debriefing scripts, and it will be made clear that questions are welcome both at the delivery of the script, and at the end of the session.

**12. All the data collected from the participants is stored securely, and in an anonymous form.**

User feedback will be anonymised and stored on the University of Sussex's OneDrive shortly after the session.

## 3 Requirements Analysis

### 3.1 Background Research

#### 3.1.1 Similar Systems

Finding similar systems beyond similar projects of the same description is difficult: many different systems are in use and most at universities are bespoke implementations, whereas similar solutions in use at Secondary Schools are often whole packages and closed-source. Therefore, while finding some examples of similar systems is possible, they are only really useful for aesthetic design inspiration.

**Panasonic AVS** ('Attendance Visualization System')[9] is a solution from Panasonic System Networks, which features real time visualisation in table form, as well as exportable csv reports. This implementation is somewhat limited by only having one visualisation and designed for employers, possibly in some sort of production line.

While it is an older piece of software, it's a good point of comparison as it is so different - this project's users would have cohorts much larger than the number of active employees in AVS, and though clocking in/out isn't something a student does, their attendance in various sessions could be visualised similarly. A tabular view similar to the figure3.1 would be useful, but only when it contains data filtered by a particular parameter.

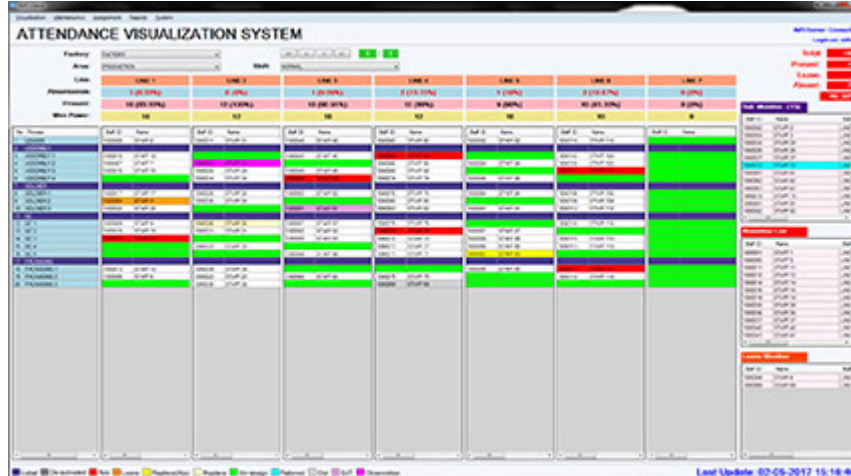


Figure 3.1: Panasonic's AVS, in a tabular view, from their promotional material[9]

**SIMS SchoolView** [10] is a bit closer to what this project seeks to achieve. SIMS is a modular school management system used primarily by Primary and Secondary schools. While the needs of these users are different to that of a University School, it's still very useful to see some examples of features

that could be incorporated into this project. It exposes both student-level data, filterable by multiple different attributes, as well as useful pre-configured dashboards for student and staff attendance 3.2.

For this project similar dashboards would be good, but ideally they should be somewhat user-configurable as user needs are likely to change over time. Filter-able table data similar to the Pupil and Staff view would be desirable features to incorporate into this project.

SchoolView

Dashboards

Views

Enterprise

Waters Edge

Period

Term 1

Surname

Any

Group By

Period

Go

Show all filter columns

Graph

Summary

Details

Status

School	Surname	Forename	Gender	Present	Late	Unauthorised	Nothing	School	Period
Waters Edge	Munster	Turic	Male	134	0	0	Nothing	English As Additional Language	Period
Waters Edge	Heaverick	Koncan	Male	152	0	0	Nothing	Ethnicity	Free School Meal
Waters Edge	Hansden	Chiko	Female	152	0	0	Nothing	Gender	Gifted And Talented
Waters Edge	Ferdinandus	Blayie	Male	152	0	0	Nothing	In Care	On Roll
Waters Edge	Baker	Flora	Female	152	0	0	Nothing	Pupil Premium	Service Children
Waters Edge	Balloy	Eve	Female	152	0	0	Nothing	Transfer Status	Year Group
Waters Edge	Cory	Max	Male	152	0	0	Nothing		
Waters Edge	Robinson	America	Female	151	0	1	Nothing		
Waters Edge	Donnan	Hubert	Male	152	0	0	Nothing		
Waters Edge	Katli	Apia	Female	152	0	0	4	0	0
Waters Edge	Coler	Somert	Male	152	0	0	4	0	0
Waters Edge	Williamson	Zachary	Male	155	0	0	1	0	0
Waters Edge	Carlson	Blairner	Female	155	0	0	1	0	0
				52968	239	115	6	339	0

Add to Dashboard

Download as CSV

(a) Overview Dashboard

SchoolView
Dashboard
Users

Emerson
Waters Edge
Period:
Term:
Surname: Any
Group By:
Period:
Show all filter columns

School	Surname	Forename	Gender	Present	Late	Unauthorised	System Status	
Waters Edge	Munster	Turic	Male	134	0	0	Nothing	
Waters Edge	Heaverick	Koncan	Male	152	0	0	School	
Waters Edge	Hansden	Chiko	Female	152	0	0	Period	
Waters Edge	Ferdinandus	Blayie	Male	152	0	0	System Status	
Waters Edge	Baker	Flora	Female	152	0	0	English As Additional Language	
Waters Edge	Balloy	Eve	Female	152	0	0	Ethnicity	
Waters Edge	Cory	Max	Male	152	0	0	Free School Meal	
Waters Edge	Robinson	America	Female	151	0	1	Gender	
Waters Edge	Donnan	Hubert	Male	152	0	0	Gifted And Talented	
Waters Edge	Katli	Apia	Female	152	0	0	In Care	
Waters Edge	Coler	Somert	Male	152	0	0	On Roll	
Waters Edge	Williamson	Zachary	Male	155	0	0	Pupil Premium	
Waters Edge	Carlson	Blairner	Female	155	0	0	Service Children	
				52968	239	115	6	Transfer Status
								Year Group

(b) Pupil and Staff Level data view

Figure 3.2: Two examples of SIMS SchoolView[10], from their promotional material.

**Sutjarittham et al: 'Tool to Access and Visualize Classroom Attendance Data from a Smart Campus'** [11] is a tool that was demonstrated at the 2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN). It was part of a research effort to investigate optimisation of classroom usage at the University of New South Wales (Sydney)[12].

While it was aimed toward optimising use of teaching spaces rather than student welfare, it's still very useful as an academic reference and for its good use of visualisations, as well as the web-based interface they have used.

## 3.2 Chosen Technologies

The chosen technology stack is a combination of ReactJS[13], Python for both a Flask[14] web server (API) and to allow for easy manipulation of data, as well as a PostgreSQL[15] database.

**ReactJS** is a JavaScript (node.js) library best known for the ease it applies to building user interfaces, as well as its multi-platform support via React Native, and plenty of packages that extend its capabilities. For these reasons it is a good choice for the user interface, and visualisations especially.

This is why React has been chosen for the front-end presentation and logic. However it cannot interact with a database on its own, which leads us onto the next level of the technology stack...

**Flask** is a Python Web Framework that is simple and easy to extend, which makes it perfect for this projects purposes as a middleman for React and PostgreSQL and there are plenty of existing implementations interfacing Flask with each.

Writing the web-server/API in Python also allows me more flexibility when it comes to data manipulation using libraries like pandas, as well as the possibility of training a predictive model on the data.

**PostgreSQL** is a object-relational database software/server, known for its feature set over the competition and wide use in web applications. In particular, it has been chosen over an equivalent technology like MySQL for it's ability to handle multiple encrypted sessions at once, which could prove useful in the eventuality the API (flask server) is overrun with requests. I've also selected Postgres (as it's commonly called) for its good community support as this will enable me to more easily deal with issues as the crop up.

All three technologies are well known in the field (so finding any issues should be fairly straightforward), and known inter-operate.

## 4 Requirements Specification

### 4.1 Mandatory Requirements

- R1 Construct a well-structured database to store attendance data, and import sample attendance data into it. The fields corresponding roughly to the current excel document headers:
- User becoming the Primary Key for the 'students' table which contains from User to Course Code
  - Level of Study being flattened to a Boolean 'isUG', True meaning Undergraduate, False meaning Postgraduate Taught
  - Course Title can be moved to a separate table 'course' that maps it to the Course Code (Primary Key).
  - Everything after Course Code will repeat each snapshot, so a table 'snapshots' with composite primary key User and 'Snapshot Date' (a Date variable corresponding to the date of insertion) will be used to differentiate them.
  - All percentage fields can be dropped as they can be easily re-calculated from their associated values.
  - Further steps may need to be taken to normalise the database as the project goes on.
- R2 Write code that presents 2D Line graphs/histograms of session attendance (as a percentage of sessions that have been attended over sessions that could have been attended) over time for the following groups:
1. Each Individual
  2. Each Degree and Stage (e.g. Computer Science - Year 3)
  3. Each Degree
  4. Each Stage
  5. Each Department (group degrees to achieve this)
  6. The School as a whole
- R3 Create a tabular view that requires at least one filter set with 'at risk students' preset, using a suitable threshold percentage of missed sessions (e.g. below 40% attendance).
- R4 Perform User Testing with the project supervisor, as well as the school's Director of Student Experience, if possible.
- R5 Create a three dimensional graph including attended sessions, time and stage, to visualise the 'fabric' of attendance for the school
- R6 Apply group filters from R2 to other fields where there is enough data to meaningfully show a trend, such as assessments turned in on time (of assessments turned in), and academic advising sessions attended of sessions scheduled.

- R7 Construct a well-structured back end (API for the database) to facilitate the other requirements data needs, with modularity and efficiency as focuses.

## 4.2 Desirable Requirements

- E1 Create a script to enable easy bulk data insertion from an excel document
- E2 Investigate applying additional visualisations, including a Nightingale Rose (as it's good for changing magnitude over time), and a Bubble Chart (change in a value over time, with a magnitude included in the size of the bubbles).
- E3 Create a Dashboard, housing useful visualisations for use by Attendance Monitoring staff
- E4 When additional data is added to the system, and new at-risk students are found, alert the user
- E5 Investigate additional filters beyond R2
- E6 Investigate if a machine learning model (from SciKit-Learn or similar libraries) can successfully predict later attendance based on earlier attendance, or likeliness of submission of assessments.



## 5 Design

The system consists of three main elements (see Figure 5.1), the underlying database which stores attendance monitoring data, the webserver that queries this database and serves API requests to the final element, the front end react application.

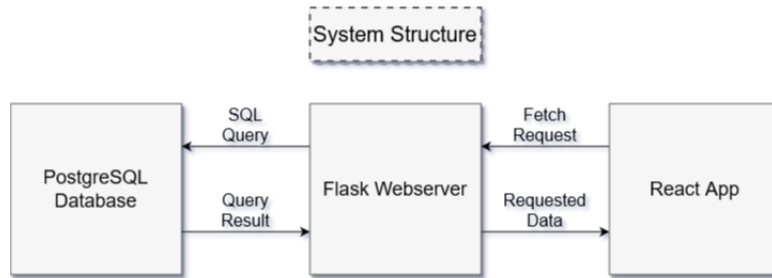


Figure 5.1: Block Diagram of the overall system architecture, centering on the Flask (Python) Webserver (Back-end) that receives requests from the React Application (Front-end), queries the PostgreSQL Database, and returns results.

### 5.1 Database Design

In line with the first mandatory requirement 4.1, the spreadsheet was split into these three tables in order to keep an up to date student object and list of courses separate from the incremental tri-weekly snapshots of each student, and overall to maintain a normalised database. It was found necessary to split these as keeping a single table like with the sample data, would be difficult to maintain, may have had issues with duplicates and conflicts, and in case the format in which data is inserted into the system were to change.

Each step mentioned in R1 was made besides level of study being flattened to a boolean in order to keep some more specificity. In addition to this, the fields 'year', 'semester', and 'week' were added to the Snapshot table to enable clearer filtering since the 'Snapshot Date' (now referred to as 'Insert Datetime') would have had more ambiguity than a concrete 'X year, Y Semester, Z Week'.

Snapshot holds the attendance data, identified by student and the represented Year, Semester and Week, with Student ID and Insertion Date/Time used as a composite primary key.

Student holds up to date information on the student (extracted from the most recently inserted snapshot), excluding their registration status, since it may be subject to change, it is held by Snapshot. Student has the primary key of an integer Student ID.

Course functions as a lookup table for unique Course Codes and Titles, with the code functioning as the primary key.

#### 5.1.1 'Student' Table

- Student ID - A unique identifier for a given student.

- (Forename, Surname) - Not yet implemented due to their absence in sample data for anonymity. Though they could be easily added to this table and relevant scripts.
- Level - The level of degree the student is most recently enrolled on e.g. 'UG' for Undergraduate, 'PGT' for Post-Graduate Taught.
- Stage - The stage of a degree the student is most recently on, e.g. 0 = Foundation, 1-5 = Year 1-5 (including an Experience/Sandwich Year)
- Course Code - The Course code e.g. 'G4010U' of the course the student is most recently enrolled on (corresponds to the equivalent primary key of the 'Course' table).

### 5.1.2 'Snapshot' Table

All values are from the time of the given snapshot, and most data points (shown here under sublists) are totals at that time.

- Student ID - A unique identifier for a given student (corresponds to the same ID on the 'Student' table).
- Year - The 4-digit year of insertion
- Semester - The semester of insertion e.g. 'Autumn', or 'Spring'.
- Week - The week of insertion e.g. 3, 6, 9, 12 (11 in modern data as Week 12 was dropped at the University of Sussex between the sample data capture and commencement of this project).
- Insert Datetime
- Registration Status - The Recorded Registration of the student at the time of the snapshot e.g. 'Registered', variations of 'PWD' (permanently withdrawn) and 'TWD' (temporarily withdrawn) corresponding to the most common reasons for withdrawal.
- Teaching
  - Sessions Count - Sessions scheduled for the student
  - Attendance Count - Sessions attended by the student
  - Explained Absence Count - The student's explained absences
  - Absence Count - The student's absences (absent with no explanation)
  - Last Session Date - The date of the student's last attended session
- Assessment
  - Assessment Count - Assessments scheduled for the student
  - Submission Count - Submissions recorded from the student
  - Explained Non-Submission Count - The student's explained non-submissions
  - Non-Submission Count - Unexplained non-submissions of assessments

- Late Period Flag - If the student's last assessment was handed in during a late period
- Last Assessment Date - The date of the student's last recorded assessment
- Academic Advising
  - AA Sessions Count - Sessions scheduled for the student
  - AA Attendance Count - Sessions attended by the student
  - AA Explained Absence Count - The student's explained absences
  - AA Absence Count - The student's absences (absent with no explanation)
  - AA Not Recorded - Unrecorded attendances
  - AA Last Session Date - The date of the student's last attended session

### 5.1.3 'Course' Table

This table functions mostly just to look up the relevant title of each course.

- Code - The standard alphanumeric code of each course represented in the data
- Title - The name of the course e.g. 'Computer Science'

### 5.1.4 Table Relationships

Both the Snapshot and Course tables relate to the Student Table, with 1-1 relationship between Student and Course, and naturally, a 1-Many relationship between Student and Snapshot.

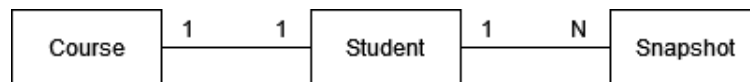


Figure 5.2: An Entity Relationship Diagram of the three aforementioned tables.

## 5.2 Back-end Design

Ease of maintenance in the form of modularity was a key consideration for the project when deciding how to go about the whole project. A custom API was chosen for this reason. By making the API and front-end separate systems with a single shared 'language' of endpoints, one can be replaced with minimal to no modification to the other. So the front-end could be dropped in favour for any preferred GUI type, including mobile or native app implementations, so long as the API is maintained; or the API could be rewritten in another language or python web framework so be it that it fully or closely maintains the functionality of the original.

A RESTful (Representational State Transfer) model was chosen as REST is a standard in modern API design and as it's features of statelessness, uniformity, and cacheability (especially for larger sets of data) stood out as very desirable features.

The four main principles of REST[16] and how they apply to this project:

- Client-Server Model / Separation of Concerns (5.1.2): Fits into the projects modularity goals in making the User Interface more portable and simplifying server scaling, as well as the need for centralised storage and management of attendance data for accuracy and timeliness.
- Statelessness (5.1.3): Storing and processing based on states would be inefficient and frankly unneeded for the general use-case of reading API data for display on a web-page.
- Caching (5.1.4): By caching large requests, especially those requiring calculations, the callback time of the front-end can be decreased with minimal to no effect on data accuracy (the larger the dataset, the less up to date changes on the small scale will effect the overall number), though keeping cache expiry times low is still very important for maximising timeliness and accuracy of returned data.
- Uniform Interface (5.1.5): Using a standardised interface formats like JSON increases interoperability and platform independence, and allows for most of the aforementioned modularity.

## Types of Endpoints

For the endpoints there were a few main considerations:

- Clarity and Usability
- Powerful Filtering
- Efficiency

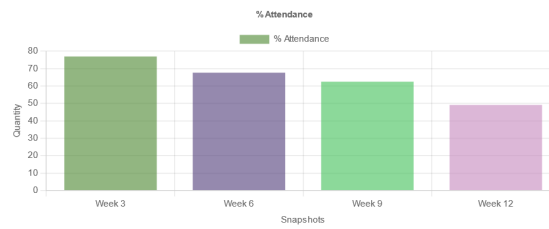
With these in mind, three types of endpoints for different data needs were settled on:

- Direct Endpoints that give exactly what's queried for based on some explicit parameters e.g. 'Students by Level'. These being for all three tables.
- Aggregate Endpoints which return aggregated counts and averages based on pre-defined groups of multiple snapshots. These being exclusively for Snapshots as that's where the quantitative attendance data is.
- Filterable Endpoints that are looser than direct endpoints and allow for powerful filtering of data based on multiple optional parameters, as well as naive search ability through the SQL 'Like' clause. These being for all tables.

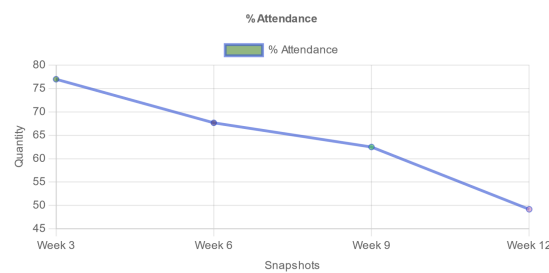
## 5.3 Front-end Design

### 5.3.1 Visualisations

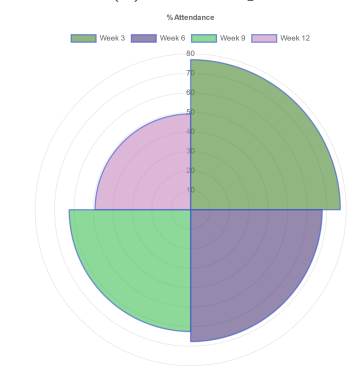
In investigating the usefulness of different visualisations three main charts were landed on (see Figure 5.3) that work best in different scenarios.



(a) Bar Graph



(b) Line Graph



(c) Polar Area

Figure 5.3: Averaged Percentage Attendance for a Computer Science Course over 4 Snapshots (1 Semester). Shown on several types of chart as part of the investigation.

Bar Charts are best used when there is only one data set being displayed, and with data that is very quantitative since they're intuitive to read from the top of bars to the axis but readability rapidly declines with large numbers of bars or when bars become stacked together horizontally or vertically (see Figure 5.4a). They're also reasonable for displaying percentages which are a key type of data for this project due to attendance percentages.

Line Charts are great for comparing multiple data sets, but suffer from readability issues due to their small single points not necessarily clearly lining up with the axis, though they're amazing for representing trends especially between different sets of data. Because of this, they're pretty atrocious for reading percentages and would not be very useful at a quick glance, especially with dense data (see Figure 5.4b).

One chart found in investigating Chart.js[17] was Polar Area, which due to it's circular form, is very useful for visualising percentages via the magnitude of different portions (in this case, quadrants, for each snapshot). They are not so useful for other types of data but excel in quick and more detailed readability for percentages due to their circular axis in addition to their clear distinction of data points (see Figure 5.4c).

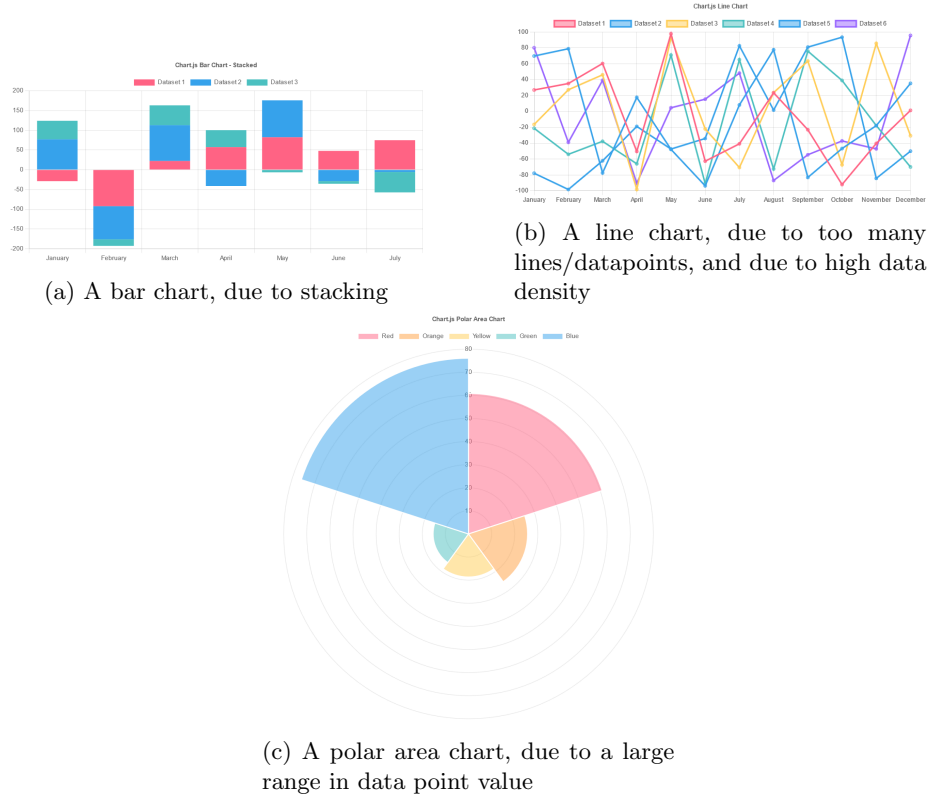


Figure 5.4: Examples of Poor Readability

Due to the aforementioned strengths and weaknesses of each, polar area charts were used for quick glance metrics like attendance percentage over time, lines for comparing multiple (often correlated) pieces of data, and bar charts to clearly show single data points over time.

### 5.3.2 User Interface

For the User Interface (UI) (See figure 5.5), lightweight, clear, and understandable design was aimed for everywhere reasonable. This is visible in the navigation bar, it's styled much like most simple websites with a high contrast dark grey to the light background making it clearly distinct from page content, as well as with the highlighted link for the currently selected page; both making navigating the different pages easy and intuitive as the user should be familiar with such design as it's common on the web.

Chart colours are selected by hashing the data-point's (in this case, week) name

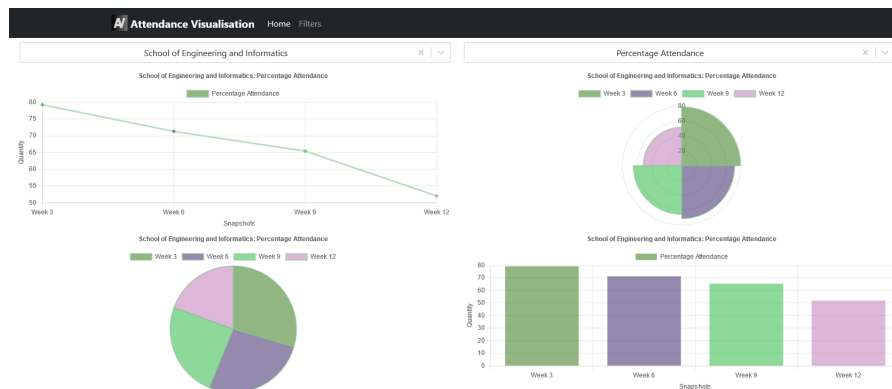


Figure 5.5: An Example of the User Interface: the 'Home' page.

to ensure they're visually distinctive.

The filtering page was designed with a fairly intuitive drop-down box to select a table to query, and then having specific forms for each table appear, with each key attribute being select-able and searchable (see figures 5.6, 5.7).

Course

g40

G4010U

G40F0U

Figure 5.6: An example of a searchable filter, in this case the Course Code on the Student form.

Table

Students

Student ID The student's unique identifier

Level Type of degree e.g. Undergrad, Postgrad

Select... | v

Select... | v

Course Code The unique identifier for their course

Stage 'years into' the degree

Select... | v

Select... | v

Filter

(a) Student Filtering UI

Table

Snapshots

Student ID The student's unique identifier.

Year

Select... | v

Select... | v

Registration Status If the student is registered or withdrawn for a specific reason.

Semester (Autumn or Spring)

Select... | v

Select... | v

Week (1-12)

Select... | v

Switch your view once you've set a filter

View as Table

Filter

(b) Snapshot Filtering UI

Table

Courses

Course Code The unique identifier of the course

Course Title The name of the course

Select... | v

Select... | v

Filter

(c) Course Filtering UI

Figure 5.7: Filtering Page UI Forms



## 6 Implementation

This section details the implementation of this project along with the challenges encountered during.

### 6.1 Version Control Issue

One challenge that effected the project at large involved version control. Initially the project was being synchronised via OneDrive, this quickly became inadequate for the PostgreSQL Server instance that wouldn't run away from its home computer (more on this in Section 6.3.1). This issue continued to effect the earliest experimental scripts for manipulating the sample data, and became a major hurdle due to many small dependency files, especially those with the '.cpython' file extension: there were easily thousands of tiny files that OneDrive was trying to synchronise whenever they were modified. This was easily fixed by moving the project into git and GitHub[18], but posed a new problem of how to create a new environment when running the project on a different computer. So a short but reliable installation guide was written, and was referred back to and used multiple times over the course of this project. This can be seen on the project's GitHub page [19].

### 6.2 Debugging between three Subsystems

Another challenge that was encountered a lot during development was just the complexity of having three different systems that could be responsible for different and sometimes compounding behaviour. When adding new features it was common for errors to ambiguously stem from either Flask or React, usually to do with formatting differences in the JSON, and often errors were due to a delay upstream (e.g. The Database being overwhelmed with requests, locking up the API, and resulting in timeouts for the React front-end).

### 6.3 PostgreSQL Database

#### 6.3.1 Setup

The first step in implementation was to construct the PostgreSQL (also referred to by just 'Postgres') database.

At first a simple approach was taken, running the standard PostgreSQL Windows Server [15] and trying to develop based around it, using pgAdmin (Postgres' administration and development tool[20]) to access and debug stored data . It worked fine on a single computer, but with the development need of using multiple computers (e.g. University Lab Machines, Personal Laptop and Desktop), and backing up the saved data, the University OneDrive was used to attempt to sync the server's files. This worked to sync data but the copied server files would not run away from their installation computer, and the thousands of small temporary files being created and synced across systems - these made syncing last for hours, which was not sustainable for long-term development.

After some frustration trying to force this setup to work, the project was changed to using ElephantSQL[21], an online 'PostgreSQL as a service' provider. ElephantSQL had a low-scale free tier, which provided more than this project

would need in terms of resources. This greatly simplified the development use of a postgres database as it was accessible from anywhere and thanks to the simplicity of Postgres URIs (Universal Resource Identifier, see Figure 6.1), a single URI could store the server address and credentials in one line of a configuration file. This significantly decreased the complexity of using a PostgreSQL server, with minimal downsides. This was because ElephantSQL very infrequently took down the instance for maintenance, though having to debug via their website was slightly more difficult than if it had been done in pgAdmin.

```
postgresql://<User>:<Password>@<Web Address>/<Database>
postgresql://me:myPassword@postgres.com/myDatabase
```

Figure 6.1: The format of a PostgreSQL connection URI, and a simple example URI.

### 6.3.2 Integration with Back-end

The second major hurdle was getting sample data from an excel file into the database. This will be discussed more in Section 6.4.1.

In general, the Flask Back-end connects to the Postgres Server with User and Password credentials stored in a configuration file ('config.yaml'), and uses its own models created with the SQLAlchemy module[22] to create the Student, Snapshot, and Course tables on the database, along with their relationships.

SQLAlchemy has been used to configure the back-end to maintain a connection with the database and a current session by which queries are sent and executed. For example, data is **selected** from the database whenever a non-cached endpoint receives a request, and **inserted** or **updated** whenever properly formed sample data is run through the excel import script. An example of one of these queries can be seen in Figure 6.2.

```
db.session
.query(Student)
.filter_by(student_id=43437412)

SELECT *
FROM Student
WHERE student_id = 43437412
```

student_id	level	stage	course_code
43437412	PGT	1	G5Q32T

Figure 6.2: An example SQLAlchemy statement [db being an SQLAlchemy Object], the equivalent PostgreSQL statement, and an expected response [formatted as a table for readability, as it would actually be returned as an SQLAlchemy Query Object].

## 6.4 Flask Back-end and API

The Back-end/API is the centre of the system. API requests come in from the React Front-end and are served via the Back-end's own queries on the PostgreSQL Database, sometimes caching the results if requests come in soon enough. Data is imported into the database via the back-end, and rearranged into the aforementioned tables on the way (see Section 5.1).

### API Python Files

- `api.py`: The main script, this is where the Flask Application is created and configured, where SQLAlchemy Models and API Endpoints are defined. This has two main sections, the first is configuration and database models; the second being the endpoints and their associated helper methods.
- `helper_methods.py`: Contains several helpful methods which were separated from `api.py` to aid readability and to avoid a monolithic application. It also contains Marshmallow[23] Schema which make it easier to strictly serialise query results.
- `excel_import.py`: Contains a single method `excel_to_db` that takes a filename, database object, year, semester and week as parameters and adds the data from the file (if it's an excel file) to the database assuming a connection to the database is open and maintained throughout.
- `config.yaml`: Contains the configuration details for the application.

### 6.4.1 Importing Data

All the functionality for data importation is contained in `excel_import.py`'s `excel_to_db` method.

The method loads the local excel file as a Data-Frame with the Pandas module[24], normalises the data, and adds or updates Students, Courses, and Snapshots depending on if they already exist or not. It does this by casting attendance metrics to integers (and None types for empty cells), and querying the database to see if a row already exists to ensure it's never committing duplicate data, updating and adding Students and adding new Courses and Snapshots. Changes are committed to the database as each row is scanned.

A post-able API endpoint for importing data was planned but not completed due to the needs of other features, namely filtering. This would have allowed an authorised user to upload an excel file on the front-end and it be posted, received and run through `excel_to_db` via a single API endpoint method. This would have enabled easier and faster administration of the system via the front-end.

### 6.4.2 API Endpoints

RESTful API endpoints were created with help from the Flask-RESTful module[25] which adds classes such as 'Resource', 'Api' and a Request Parser ('reqparse') that were useful in structuring the REST API. Resource was useful for easily defining the endpoints as RESTful resources, the api class made it easy to assign specific URLs to these resources, and the request parser enabled more variable endpoints (in this case the 'Filterable Endpoints' we defined earlier).

All endpoints utilise a helper method to reformat the query object into a JSON serialise-able Python Dictionary and produce the format expected by the front-end (key: a unique identifier, value: the expected row).

**Direct Endpoints:** The first endpoints created were the aforementioned 'direct' endpoints, named as such as they're simple and intended to be used for specific circumstances. They focus on a single table and field from that table to search by, e.g. Student by Course Code, and returning the results. When Filterable endpoints were created, it was debated whether or not to keep these more naive ones, but it was ultimately decided to maintain them as it is very clear to any API users what they do and how they filter, and are therefore both valuable in terms of having very concrete functionality that should not change, and in being a starting point for someone learning the system. For an example of a direct endpoint for 'Student', see Figure 6.3.

```
{
  "student_id": 43437412,
  "level": "PGT",
  "stage": 1,
  "course_code": "G5Q32T"
}
```

Figure 6.3: An example JSON response containing all the Student table fields, from direct endpoint '/api/student/student\_id/43437412'. This is the same data as in Figure 6.2, for comparison.

**Aggregate Endpoints:** The need for aggregate data (sums, averages, minimum, and maximum values for groups of students) motivated the development of aggregate endpoints, sharing the common method `get_aggregate_data` which takes a list of student IDs in the relevant group and returns the aforementioned aggregated values for each snapshot week found (in practice with the sample data this is weeks 3, 6, 9 and 12). For Courses, Stages, and Schools (all the sample data as it's all for just one school) this was simple enough, but for Departments, a manual list was needed to separate one department's courses from another. Using a list of the values that can be summed, averaged, and minimum and maximum values found for - named 'attributes', `get_aggregate_data` queries all Snapshots for the given student and then runs a query for each attribute which returns its minimum, maximum, average and sum values for all Snapshots in a given week. An example of the aggregate data returned by these endpoints can be seen in Figure 6.4.

**Filterable Endpoints** were added to achieve one of the main aims for the system: being able to filter students based on their attendance, specifically the percentage of the time they have been attending, though this functionality didn't directly make it into the final product. Nevertheless, filterable endpoints enabled Students, Snapshots, and Courses to be filtered by any of their fields (though metrics are excluded for Snapshot), as seen in Figure 5.7.

Filterable endpoints also provided a naive search function with the ability to

```

"teaching_attendance": {
  "3": {
    "min": 0,
    "max": 42,
    "avg": "15.2750737463126844",
    "sum": 20713
  },
  "6": {
    "min": 0,
    "max": 40,
    "avg": "13.6119733924611973",
    "sum": 18417
  },
  "9": {
    "min": 0,
    "max": 43,
    "avg": "12.9168522642910171",
    "sum": 17399
  },
  "12": {
    "min": 0,
    "max": 32,
    "avg": "7.3040892193308550",
    "sum": 9824
  }
}

```

Figure 6.4: An example JSON array of Teaching Attendance from the Aggregate Endpoint '/api/aggregate/school' for the entire School of Engineering and Informatics.

use the 'like' keyword exclusive to Postgres which allows for pattern matching including wildcards. This would allow searching for strings that include a given sequence by putting wildcards either after the string or both before and after it. Using the 'like' keyword also added a level of flexibility since it's unlikely while searching that someone would know the exact name or value of a field exactly as it was entered into the database. Figure 6.5 demonstrates an example filterable endpoint on the Student table.

The Filter Options Endpoint is unique in having been added purely for front-end UI, it simply returns the list of possible filters depending on which table name it has been passed. This was needed as the select boxes on the Filter page needed to be populated with select-able options for each filterable field e.g. Student ID, Course, Level and Stage for filtering Students.

### 6.4.3 Caching

Caching was achieved using the Flask-Caching module[26] to do most of the heavy lifting, only requiring some configuration and additional cache statements in each API Resource class. Flask-Caching holds a copy of a given cached

```

{
  "43006762": {
    "student_id": "43006762",
    "level": "UG",
    "stage": "3",
    "course_code": "G4010U"
  },
  "43006800": {
    "student_id": "43006800",
    "level": "UG",
    "stage": "3",
    "course_code": "G4010U"
  }
}

```

Figure 6.5: An example JSON response from the endpoint (including arguments) `/api/filter/student?level=UG&stage=3&course_code=G4010U`.

This is a filter on the Student table, and presents students who are undergraduates, in the third year of their degree, and are doing Computer Science MComp (Integrated Masters).

endpoint's response for as long as described in the cache statement, or for the chosen default time-out period (in this case 60 seconds).

To avoid tying up the database or back-end at scale (e.g. with excessive reloads or just many concurrent users), every endpoint (excluding filterable ones) is cached with a keep-alive of 60 seconds, and more computationally intensive endpoints are cached relative to the assumed size of the data set being computed: in the case of the Aggregate Endpoints this is 150 seconds for a whole Stage (also commonly referred to as a 'year group'), 300 seconds for a whole department, and 600 for an entire school. Since these data sets are increasingly large, the difference in calculated values from a cached version to a slightly more up-to-date version, is in practice likely to diminish exponentially, especially for the main target of the Aggregate endpoints: Averages. This seemed to be a good justification for caching these fairly computationally intensive calculations, especially since they had the potential to tie up not just the back-end, but the database too. This also served to make the front-end seem snappier as it didn't always have to wait on the database to calculate and pass back results via the API.

#### 6.4.4 Integrating a Flask / React Project

Getting two different web frameworks to work around each-other within the same cohesive project is not trivial. This project utilised parts of a blog post (by Miguel Grinberg, Software Engineer and Technical Writer[27]) to structure and configure the code-base initially.

**Structure:** The main project directory is the create-react-app[28] directory, and the Flask API is contained in the 'api' sub-directory.

**Configuration:** The most significant part used from this post was how to configure the two to work together, which consisted of adding a script called 'start-api' to the React App's package.json file (used by npm, the JavaScript package manager, to keep track of project metadata) which would start the Flask Web-server inside of its own virtual environment, and a proxy that redirects traffic the React Application cannot serve to the Flask Web-server. The former required a Flask environment be configured (using the Python module python-dotenv [29]) so that flask knows what to run; The latter was achieved by having the two on two different ports, 3000 for React, 5000 for Flask.

## 6.5 React Front-end

The React Front-end interacts only with the Flask API, but is the home of all User Interfaces for the system. Most data presented on the front-end is directly extracted from API calls to the back-end. The React Application is structured akin to the default 'create-react-app' project[28] but with the key difference of having 'helper' and 'routes' sub-directories. 'helper' files house common functions and 'routes' files are the distinct pages that utilise them.

- helper
  - chartHandling.jsx - Functions that extract and rearrange JSON Objects into Objects acceptable by Chart.js[17], as well as giving each datapoint (in this case often each 'Week') a distinct colour value with the 'Color-Hash' package[30]
  - fetchApiData.jsx - A function that returns JSON data when given an endpoint to target
  - roles.jsx - Helper functions for planned role based access control
- routes
  - home-page.jsx - All functions behind the homepage besides helpers
  - filters-page.jsx - All functions behind the filters page besides helpers
  - root.jsx - Navigation bar and root React-Router route, automatically redirects to the home page. And because relative paths were used, navigation doesn't require page reloads.
  - error-page.jsx - Simple error page to show when root.jsx catches an error
  - **Planned pages: (hidden from navigation)**
  - *data-page.jsx - Data Management page (edit, upload, and delete data)*
  - *users-page.jsx - User Management page (add users, roles, general administration)*
  - *login-page.jsx - User authentication, password reset etc.*
- App.jsx - The Application itself, uses React-Router[31] to display different pages based on the provided path.
- index.jsx - Renders the application

### 6.5.1 Homepage/Dashboard

The homepage allows users to filter aggregate data for a big-picture view of whichever metric they may be interested in. This is achieved with two drop-downs powered by React-Select, though they could have easily been vanilla and functioned the same. The first drop-down allows the user to select a Student Group, the available options being based around the Aggregate Endpoints: The School, Departments, Stages, and Courses. The second drop-down allows the user to select a metric which includes re-calculated percentages from the sample data (Percentage of Attendance, Percentage of Assessments Submitted, and Percentage of Academic Advising Attendance), and the numeric metrics from the Snapshot table.

At the moment this 'dashboard' displays the same data on all four charts, but this could be made further configurable in future and it is beneficial in exploring visualisation to see how the same data can look slightly different in different mediums. Example of the home page in use can be found in Figure 6.6, or earlier in Figure 5.5.

### 6.5.2 Filtering Page

The filters page allows users to filter through table data using some preset fields for each. This is achieved with a few key React 'hooks' (functions that allow developers to 'hook into' state and other features).

The main hook 'FiltersPage' holds the state of another hook 'TableFilters', and switches which 'TableFilters' appears based on the Select drop-down at the top of the page. TableFilters holds the most of the logic for the page, rendering and handling the different forms for each table, and keeping track of data needed for different inputs for the 'View' hook. View is a fairly naive hook that just returns data passed to it in the type it has been asked for. At the moment this is just 'line' for line charts and 'table' for tables but it can be expanded easily with need.

The resulting UI can be seen in Figure 5.7, and an example use-case in Figure 6.7.

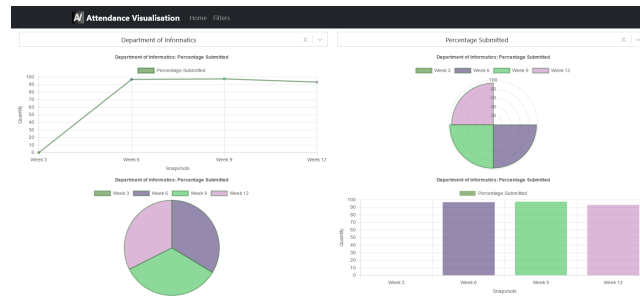
### 6.5.3 Visualisations

Visualisations were created using the package react-chartjs-2[32] (which creates React Components for Chart.js[17]).

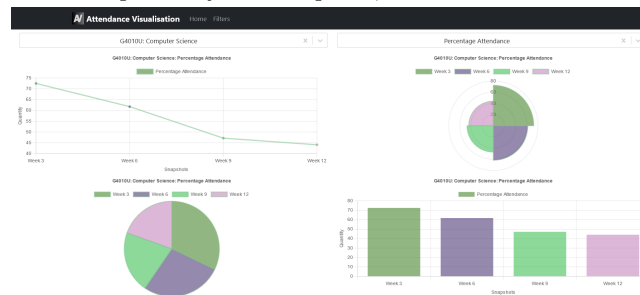
Several functions from helper file 'chartHandling' were used to extract and process data from each data-set and return it in the format mandated by Chart.js (ending in 'Data'), as well as generate different configured options allowed in Chart.js (and its plugins) to customise the appearance and functionality of the charts (ending in 'ChartOptions'. These included:

- 'ExtractChartData', which extracts data from multiple snapshots of the same individual,
- 'ExtractAggregateData', which extracts a specified aggregate data value (Minimum, Maximum, Sum, and Average values) for a student group

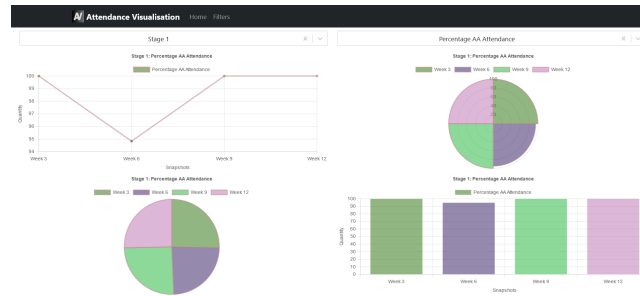




(a) An example of the homepage in use, displaying charts of the percentage of assigned assessments submitted within the Department of Informatics. Note: there is no data shown for Week 3 as there were no assessments assigned in the range of time depicted by this Snapshot, Weeks 1-3.



(b) An example of the homepage in use, displaying charts of the percentage of scheduled teaching that was attended by students on the Course G4010U 'Computer Science'.



(c) An example of the homepage in use, displaying charts of the percentage of scheduled and attended academic advising sessions for all Stage 1 (First Year) students in the School of Engineering and Informatics.

Figure 6.6: Examples of the Homepage in use.

- 'PercentageChartData', which does the same as 'ExtractChartData' but returns the data as a percentage constructed of a given numerator over a given denominator which can be any two metrics from the 'Snapshot' table.

Table

Snapshots

Student ID The student's unique identifier.

Year

Registration Status If the student is registered or withdrawn for a specific reason.

Semester (Autumn or Spring)

Week (1-12)

Switch your view once you've set a filter

View as Line Chart

Select...

Filter

student_id	year	semester	week	insert_datetime	registration_status	teaching_sessions	teaching_attendance	teaching_explained_absence	teaching_absence	teaching_last
42003972	2017	Autumn	3	2023-03-30 22:56:20.897570	REGISTERED	21	5	2	14	2017-10-16
42806822	2017	Autumn	3	2023-03-30 22:56:28.940176	REGISTERED	18	16	0	2	2017-10-16
42003972	2017	Autumn	6	2023-03-30 23:00:20.078747	REGISTERED	11	7	2	2	2017-11-03
42806822	2017	Autumn	6	2023-03-30 23:00:27.625997	REGISTERED	12	11	0	1	2017-11-06
42003972	2017	Autumn	9	2023-03-30 23:04:05.031943	REGISTERED	11	4	0	7	2017-11-24

Figure 6.7: An example use of the filtering page, with the following filters set: Student IDs '42003972' and '42806822', Registration Status set to 'REGISTERED', the Year 2017, and the Semester 'Autumn'. Note: this image only contains a subset of the data on the page for demonstrative purposes.

- 'PercentageAggregateData', which calculates percentages from aggregate data similar to 'ExtractAggregateData' and 'PercentageChartData'
- 'LinearChartOptions', which returns appropriate chart options for Bar and Line charts
- 'CircularChartOptions', which returns appropriate chart options for Polar Area and Pie Charts

The charts presented in the project are also given hashed background colours to make sure each data-point can be distinguished.

Since the react-chartjs-2 components only need their 'data' and 'options' props/parameters filled, a mixture of these functions alongside Chart.js fulfil, for now, the project's charting needs.

## 7 Evaluation

### 7.1 Testing

In the course of development, the project's three systems were tested using several methods:

- The API's JSON Output was tested by directly accessing the relevant endpoint in a browser (e.g. at 'localhost:5000/api/aggregate/student/'), and if there wasn't an obvious difference, run through a difference checker (e.g. Diffchecker.com [33]) against a similar working output while ignoring what should be different.
- The Database's values and Backend-Database interactions were tested by querying relevant tables on ElephantSQL's browser using standard PostgreSQL statements, for example:  
`'SELECT student_id, date(insert_datetime), COUNT(*)  
FROM "public"."Snapshot"  
GROUP BY student_id, date(insert_datetime)'` was used to get the number of snapshots per student to determine whether there were any missing or duplicate snapshots in the upload process, since in the sample data, every Student should have 4 Snapshots associated with them.
- The Back-ends internal functions were tested using a mixture of print statements and Flask's own debugging messages (IDE-specific debugging features were not used as the Flask app was being run directly from the console, not via an IDE)
- The Front-ends internal functions were tested by logging to the browser console the results of actions and were also used to determine where in the functions had actually been reached, as well as what was right/wrong about produced values.

Features were regularly tested throughout development, especially around implementation of similar features which could effect others. Usability is tested in Section 7.2 from the Front-end User Perspective. In general, in spite of some bugs and the lack of a limit on the quantity of data that can be returned, the project functions as specified.

### 7.2 Usability Study

#### 7.2.1 Methods

Before beginning, the participants are briefed on the project by being provided a copy of the Introduction to this report<sup>1</sup>, and all participants agreed to perform user testing with no incentives, and were not exposed to any greater risks than in daily life as the system was running locally on a laptop and were not taken to another environment to participate. As all participants were fellow students at the University of Sussex, it is reasonably assumed that they are above the age of 18, and are capable of consent.

This study was composed of two rounds:

**Round 1.** A preliminary round of giving participants access to the system with no direction beyond asking that they explore and see what they think of it, and observing how they get on (though any participants questions about the system are answered to the best of the facilitator's ability). The participant continues for as long as they think is reasonable to explore the system, but in practice this doesn't last much longer than 5-10 minutes. This is followed then by asking for their feedback as to how usable the system is and any improvements that could be made.

**Round 2.** A more focused round where the participants are individually given access to the system with specific tasks to attempt in the session. This is intended to garner more specific feedback about different workflows on the system.

Every participant was given the following 10 tasks to attempt in Round 2, and asked to rate how usable the application was for each task on a scale of 1 to 10:

1. (On the home page) View the Percentage of Attendance for the School of Engineering and Informatics
2. View the Percentage of Assessments Submitted for the Department of Informatics
3. View the Percentage of Academic Advising Attendance for the Course 'G4010U' (Computer Science MCOMP)
4. View the Teaching Absence for the Course 'G5001U' (Computer Science BSc)
5. Navigate to the 'Filters' page
6. Select each table in the top drop-down: Student, Snapshot, and Course
7. Select Student, and filter for Undergraduates and Postgraduates in the Second year of their degree
8. Select Snapshot, and filter for Student 43402540
9. Select Course, and filter for Courses with the title 'Computer Science'
10. Navigate back to the 'Home' page

Between the two rounds of usability testing there was some time between them put to use to improve the usability of the system, namely requests for:

- Flavour text on the filter page - making it clear what things are and what they do
- Having more charts on the homepage, specifically a bar chart
- Making the homepage data-sets select-able and less basic

Fixes for these were implemented before Round 2, and are present in the final product.

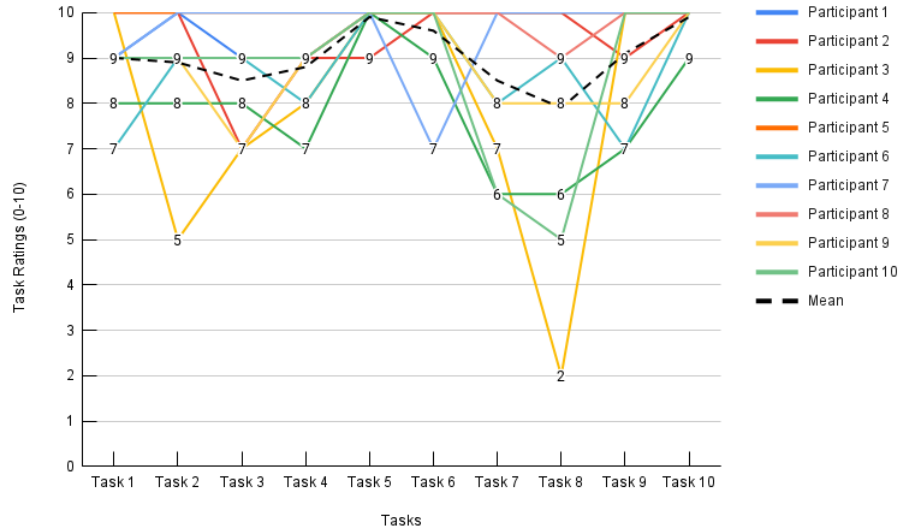


Figure 7.1: A line chart depicting the resulting ratings from Round 2. Full data is available as a table in Appendix A.

### 7.2.2 Results and Evaluation

Figure 7.1 shows the individual and mean average ratings for each task. The figure serves well as a general 'heatmap' of bugs and failed user interactions. Ratings below 10 tend to be from buggy or non-obvious behaviour, notably from drop-downs (Select boxes) being unclear. There is a noticeable dip in the mean for Task 8, which is down to the buggy behaviour of the Snapshot filter if nothing has been selected (all Snapshots are returned, since a limit was not set in development due to more demanding features), as well as the line chart not being very robustly implemented. The simple navigation based tasks (5 and 10) were very well received, probably due to the ease of clicking a single hyperlink, whereas more complex and non-obvious actions such as filtering and selecting student groups and data-sets were received worse. This was especially true for Task 8 as already mentioned, since there was still some lack of clarity in the form(s).

	Textual Response
1	Deselecting a dataset specifically gives an error. Choosing the dataset before the student group also requires double clicking the dataset to register the change. Filters are completely fine, overall very nicely designed UI. :)
2	I think the navigation links can be slightly bigger or with a larger contrast to the page, so they can be more easily identifiable. I also think the options on some of the dropdowns could be alphabetically sorted for easier search and be less ambiguous in content. Also it could be more obvious that you could search by typing on some of them for quicker search.
3	Following HCI principles, making the data more visible when desired, and hiding it when unnecessary might be helpful in making the user interface less crowded and more intuitive.
4	The scrolling feature in the students table menu might be modified as the scrolling process might seem ambiguous for a certain group of users. Instead, it might now show available options before the first digit is put or completely replaced by the manual typing option.
5	when typing in student group into the home page, it would show data for that group, but then if the same group was selected instead by selecting it from the dropdown, the data changes or is then displayed differently.
6	mostly good to use, some aspects could potentially be labelled better. potentially load in the full student dataset on the homepage as the default and also a clear all button maybe. thank
7	I think you can add units for dataset when choosing it. like teaching lesson (per lessons as unit)
8	Buggy. Live update rather than explicit buttons would be nice. Points on graph are small and therefore difficult to hover over for exact figures on line graph. Would be nice if it brought up the information based on proximity rather than actual hover over. Hi Paul! How's Dan doing in his Diss?
9	I think it worked really well and serves its purpose. Only minor things that could be improved. It would be good to have the ability to sort results and being able to search fields being clearer
10	The drop down menus are a very light grey and hard to spot at a glance. but there are very few places to interact so it is still easy to find what i'm looking for. The dropdown boxes weren't intuitive that i could select multiple options. Some bugs but overall the style of the site is very clean and easy to navigate with data visualised well.

Table 1: Participant Number versus the participants textual responses.

While the quantitative data is great for finding trends, the textual responses gathered from Round 2 were far more useful for finding what participants actually thought of the user interface. The above table lists these. Overall, these responses were overwhelmingly positive, though there were a fair few User Experience problems and bugs identified by the participants.

A few common complaints are:

- Ambiguity in drop-downs, in terms of the ability to select multiple options, being able to type to search the options, the content (from the sample data-set) listed, and the meaning of fields
- Navigation links are small and have too little contrast to the navigation

bar

- Unsorted data from the API / Filtered data in tabular form should be sort-able by each field
- Filtering should update on changes to the form, not an explicit button
- Ambiguity in the meaning of data-sets represented on the Home page

These responses are well justified as the user interface does suffer from these problems, in part due to the way it was developed: incrementally based on need rather than properly planned along with user experience. This was done due to the React application being the final element of the project and the API needing more work than was previously estimated.

To overcome these problems, more work would need to be put into planning out the user experience and making more use of Human-Computer Interaction rules, and usability testing would need to be conducted again once they appeared to be solved. To fix the more data-related problems in particular, the database could be adjusted to include more data such as what degree type (e.g. 'BSc', 'MComp') a course is, in order to avoid ambiguity on that select option, and API endpoints and Database tables designed more tightly around what is needed in the user experience rather than the user experience having to fit pre-existing system structures.

## 7.3 Evaluation Overview

Of the mandatory requirements specified in Section 4, 4 were met, 1 was partially met, and 2 were not met. And of the desirable requirements, 3 were met, 1 was partially met, and 2 were not met.

### 7.3.1 Mandatory Requirements

Requirement 1 was satisfied, though the flattening of Level of Study to a boolean 'isUG' was reversed to improve user interface clarity, as the strings 'UG' and 'PG' in a selection box are far more obviously Undergraduate and Postgraduate than a checkbox or other boolean control would have been.

Requirement 2 was satisfied, as 2D line graphs were plotted for each individual, degree and stage (though not shown in the final product, the functionality exists for this in the API), degree alone, stage alone, each department, and the school as a whole (though not as histograms as the data spread was not wide enough to justify binning data, and bar charts were used instead).

Requirement 3 was not satisfied due to commitment to other features, though could be implemented with very little modification to the API and a single additional Front-end page.

Requirement 4 was partially satisfied as user testing was performed with peers but not the project's supervisor or the Director of Student Experience.

Requirement 5 was not satisfied due to commitment to other features and due to the charting library selected not supporting 3D charts, though with a more granular charting library, it could be achieved with minimal modification to the front-end and no modification to the API or database.

Requirement 6 was satisfied as the mentioned data-sets were select-able on the home page of the React Application, and were even the subject of two of the Usability Study's tasks (Tasks 2 and 3).

Requirement 7 was satisfied as a well-structured back-end API was constructed and facilitated the other implemented requirements data needs, and displays modularity and efficiency (e.g. caching) in design.

### **7.3.2 Desirable Requirements (Extensions)**

Extension 1 was satisfied as it was a first step toward constructing the overall project, since an API could then be implemented and tested on data from the database.

Extension 2 was partially satisfied as bubble charts were experimented with initially, but in having 3-dimensional data, they're inherently difficult to read and by having the magnitude of circles depicting a quantity, preciseness is sacrificed greatly. This extension was only partially satisfied also since Nightingale Rose visualisations specifically were not explored, though the Polar Area chart was found and utilised in this project.

Extension 3 was satisfied as a dashboard (the homepage) was created and it could reasonably be argued to contain at least somewhat useful visualisations for Attendance Monitoring staff.

Extension 4 was not satisfied as notification functionality, let alone user account functionality, was not implemented.

Extension 5 was satisfied as additional filters, in particular the filters shown on the filters page, were investigated and found to be useful for retrieving table data.

Extension 6 was not satisfied as other functionality, specifically the API, was prioritised over investigating a machine learning model. And the sample data would likely not be a large enough data set (not in terms of number of students, but snapshots captured) to accurately train such a model.



## 8 Conclusion

### 8.1 Objectives

For reasons stated in Section 7.3, primary objectives 1, 2, 3, 5 and 6 were achieved, and primary objective 4 was partially achieved since user testing was not done with relevant staff but rather with students. Extensions 1 and 3 were achieved and extensions 2 and 4 were partially achieved, as respectively, the application was technically user configurable but not in a lasting state (i.e. with saved configurations) and it did not include multiple dashboards, and as meaningful statistics were found but they were percentages and therefore largely based on quantities.

Though some individual objectives were not completely met, in general the objectives were met over the course of this project, and its motivations fulfilled to some extent.

### 8.2 Retrospectives

In hindsight, a few things could have been done differently in this project:

- Faker[34], a Javascript library for generating fake but realistic data for development. Experimenting with and potentially adding this into the project may have been invaluable in both making the system more realistic by having more than a semester of data, and if the data was still indicative of patterns found in the sample data, may have even been useful for training the machine learning model that was un-prioritised from Desirable Requirement E6.
- A more familiar technology stack could have been used to produce more impressive results, though I am confident I would not have learnt nearly as much if this was the case, and the project still turned out well in spite of inexperience with PostgreSQL, Flask, and React. So this is a decision I would stick with for the sake of my professional development and skill.
- The hashed values used for chart colours could have been salted (an additional bit of distinguishing data added) with the name of the metric to make them more distinct between displayed metrics.

### 8.3 Further Work

To build on this project, one could try:

- In general, further refining it into a more structured product: re-designing user interfaces based on the Usability study, linking the 'excel.import' python script into a post-able endpoint to allow data upload from the user interface directly to the database via the API/Back-end, adding user accounts and administrator functionality for these as well as the data without accessing the database directly.
- Running a dedicated PostgreSQL server myself, with considerations made to appropriate configurations for the projects data and common interactions.

- Devising a deployment strategy for a more finished system, as the project was developed and tested entirely locally.
- Creating and refining a fully list of required packages, frameworks, and languages, in order to better organise dependencies and compatibility beyond the two different requirement files ('packages.json' and 'api/requirements.txt') and language versions the project was developed on.
- Improve caching to include filter endpoints, as well as overhauling endpoints to validate user input, possibly accept it for updating data, and implementing limits on the quantity of returned data .

## References

- [1] John Colby. ‘Attendance and Attainment - a comparative study’. In: *Innovation in Teaching and Learning in Information and Computer Sciences* 4.2 (May 2005). Publisher: Routledge \_eprint: <https://doi.org/10.11120/ital.2005.04020002>, pp. 1–13. ISSN: null. DOI: 10.11120/ital.2005.04020002. URL: <https://doi.org/10.11120/ital.2005.04020002> (visited on 14/10/2022).
- [2] Loretta Newman-Ford et al. ‘A large-scale investigation into the relationship between attendance and attainment: a study using an innovative, electronic attendance monitoring system’. In: *Studies in Higher Education* 33.6 (Dec. 2008). Publisher: Routledge \_eprint: <https://doi.org/10.1080/03075070802457066>, pp. 699–717. ISSN: 0307-5079. DOI: 10.1080/03075070802457066. URL: <https://doi.org/10.1080/03075070802457066> (visited on 14/10/2022).
- [3] Bernard Meulenbroek and Maartje van den Bogaard. ‘Attendance and attainment in a Calculus course’. In: *European Journal of Engineering Education* 38.5 (Oct. 2013). Publisher: Taylor & Francis \_eprint: <https://doi.org/10.1080/03043797.2013.811478>, pp. 532–542. ISSN: 0304-3797. DOI: 10.1080/03043797.2013.811478. URL: <https://doi.org/10.1080/03043797.2013.811478> (visited on 14/10/2022).
- [4] Gabrielle E. Kelly. ‘Lecture attendance rates at university and related factors’. In: *Journal of Further and Higher Education* 36.1 (Feb. 2012). Publisher: Routledge \_eprint: <https://doi.org/10.1080/0309877X.2011.596196>, pp. 17–40. ISSN: 0309-877X. DOI: 10.1080/0309877X.2011.596196. URL: <https://doi.org/10.1080/0309877X.2011.596196> (visited on 14/10/2022).
- [5] *Facts and figures : Rankings and figures : About us : University of Sussex*. en. URL: <https://www.sussex.ac.uk/about/facts/facts-figures> (visited on 31/10/2022).
- [6] *Record numbers of students take up university places*. en. URL: <https://www.gov.uk/government/news/record-numbers-of-students-take-up-university-places> (visited on 31/10/2022).
- [7] *BCS Code of Conduct for members - Ethics for IT professionals — BCS*. URL: <https://www.bcs.org/membership-and-registrations/become-a-member/bcs-code-of-conduct/> (visited on 05/11/2022).
- [8] *Research Governance : Research : School of Engineering and Informatics : Schools and services : University of Sussex*. URL: <https://www.sussex.ac.uk/ei/internal/research/researchgov> (visited on 26/10/2022).
- [9] *Attendance Visualization System - Panasonic System Networks Malaysia Sdn. Bhd. (42154-T) [PSNM]*. URL: <https://snm.my.panasonic.com/attendance-visualization-system> (visited on 03/11/2022).

- [10] *SIMS SchoolView*. en. URL: <https://www.ess-sims.co.uk/sites/default/files/2021-06/SIMS%5C%20SchoolView%5C%20-%5C%20June%5C%202021.pdf> (visited on 07/11/2022).
- [11] Thanchanok Sutjarittham et al. ‘Demo Abstract: A Tool to Access and Visualize Classroom Attendance Data from a Smart Campus’. In: *Conference: 2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. Apr. 2018, pp. 140–141. DOI: 10.1109/IPSN.2018.00033.
- [12] Thanchanok Sutjarittham et al. ‘Data-driven monitoring and optimization of classroom usage in a smart campus’. In: *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*. IPSN ’18. Porto, Portugal: IEEE Press, Apr. 2018, pp. 224–229. ISBN: 978-1-5386-5298-5. DOI: 10.1109/IPSN.2018.00050. URL: <https://doi.org/10.1109/IPSN.2018.00050> (visited on 08/11/2022).
- [13] *React – A JavaScript library for building user interfaces*. en. URL: <https://reactjs.org/> (visited on 10/11/2022).
- [14] *Welcome to Flask — Flask Documentation (2.2.x)*. URL: <https://flask.palletsprojects.com/en/2.2.x/> (visited on 10/11/2022).
- [15] PostgreSQL Global Development Group. *PostgreSQL*. en. Nov. 2022. URL: <https://www.postgresql.org/> (visited on 10/11/2022).
- [16] Roy Thomas Fielding. ‘Architectural Styles and the Design of Network-based Software Architectures DISSERTATION’. en. In: *University of California, Irvine* (2000). URL: [https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding\\_dissertation.pdf](https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf).
- [17] *Chart.js*. en-US. URL: <https://www.chartjs.org/> (visited on 19/04/2023).
- [18] *Build software better, together*. en. URL: <https://github.com> (visited on 25/04/2023).
- [19] Daniel Bates. *Attendance-Visualisation*. Apr. 2023. URL: <https://github.com/danbates1452/Attendance-Visualisation> (visited on 22/04/2023).
- [20] *pgAdmin - PostgreSQL Tools*. URL: <https://www.pgadmin.org/> (visited on 19/04/2023).
- [21] *ElephantSQL - PostgreSQL as a Service*. en. URL: <https://www.elephantsql.com/> (visited on 27/03/2023).
- [22] *SQLAlchemy - The Database Toolkit for Python*. URL: <https://www.sqlalchemy.org/> (visited on 19/04/2023).
- [23] *marshmallow: simplified object serialization — marshmallow 3.19.0 documentation*. URL: <https://marshmallow.readthedocs.io/en/stable/> (visited on 20/04/2023).
- [24] *pandas - Python Data Analysis Library*. URL: <https://pandas.pydata.org/> (visited on 20/04/2023).
- [25] *Flask-RESTful — Flask-RESTful 0.3.8 documentation*. URL: <https://flask-restful.readthedocs.io/en/latest/> (visited on 21/04/2023).

- [26] *Flask-Caching — Flask-Caching 1.0.0 documentation*. URL: <https://flask-caching.readthedocs.io/en/latest/> (visited on 21/04/2023).
- [27] Miguel Grinberg. *How To Create a React + Flask Project*. en. URL: <http://blog.miguelgrinberg.com/post/how-to-create-a-react--flask-project> (visited on 22/04/2023).
- [28] *Create React App*. en. URL: <https://create-react-app.dev/> (visited on 22/04/2023).
- [29] *python-dotenv*. URL: <https://saurabh-kumar.com/python-dotenv/> (visited on 23/04/2023).
- [30] *color-hash*. en. Dec. 2022. URL: <https://www.npmjs.com/package/color-hash> (visited on 22/04/2023).
- [31] *Home v6.10.0 — React Router*. en. URL: <https://reactrouter.com/en/main> (visited on 22/04/2023).
- [32] *react-chartjs-2 — react-chartjs-2*. en. URL: <https://react-chartjs-2.js.org/> (visited on 22/04/2023).
- [33] *Diffchecker - Compare text online to find the difference between two text files*. en. URL: <https://www.diffchecker.com/> (visited on 26/04/2023).
- [34] *Faker — Faker*. en-US. URL: <https://fakerjs.dev/> (visited on 26/04/2023).

## Appendix A Raw Usability Study Response Data

Participants' Task ratings versus Tasks:

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
P1	9	10	9	9	10	10	10	10	10	10
P2	10	10	7	9	9	10	10	10	9	10
P3	10	5	7	8	10	10	7	2	10	10
P4	8	8	8	7	10	9	6	6	7	9
P5	10	10	10	10	10	10	10	10	10	10
P6	7	9	9	8	10	10	8	9	7	10
P7	9	10	10	10	10	7	10	10	10	10
P8	9	9	9	9	10	10	10	9	10	10
P9	9	9	7	9	10	10	8	8	8	10
P10	9	9	9	9	10	10	6	5	10	10
Mean	9	8.9	8.5	8.8	9.9	9.6	8.5	7.9	9.1	9.9

## Appendix B Project GitHub Repository

Accessible at

<https://github.com/danbates1452/Attendance-Visualisation>.

## Appendix C User Testing Compliance Form

Can be found at the next page...

## User Testing Compliance Form for UG and PGT Projects\*

### School of Engineering and Informatics

### University of Sussex

This form should be used in conjunction with the document entitled “Research Ethics Guidance for UG and PGT Projects”.

Prior to conducting your project, you and your supervisor will have discussed the ethical implications of your research. If it was determined that your proposed project would comply with **all** of the points in this form, then both you and your supervisor should complete and sign the form on page 3, and submit the signed copy with your final project report/dissertation.

If this is not the case, you should refer back to the “Research Ethics Guidance for UG and PGT Projects” document for further guidance.

---

1. Participants were not exposed to any risks greater than those encountered in their normal working life.

*Investigators have a responsibility to protect participants from physical, mental and emotional harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, physical hazards or discomfort, emotional distress, use of sensory deprivation (e.g. ear plugs or blindfolds), sensitive topics (e.g. sexual activity, drug use, political behaviour, ethnicity) or those which might induce discomfort, stress or anxiety (e.g. violent video games), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback.*

2. The study materials were paper-based, or comprised software running on standard hardware.

*Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, mobile phones, and tablet computers is considered non-standard.*

3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project.

*Participants cannot take part in the study without their knowledge or consent (i.e. no covert observation). Covert observation, deception or withholding information are deemed to be high risk and require ethical approval through the relevant C-REC.*

---

\*This checklist was originally developed by Professor Steven Brewster at the University of Glasgow, and modified by Dr Judith Good for use at the University of Sussex with his permission.

*If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, the data is to be published or there are future secondary uses of the data), then it will be necessary to obtain signed consent from each participant. Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script (see Appendix 1).*

4. No incentives were offered to the participants.

*The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle. People volunteering to participate in research may be compensated financially e.g. for reasonable travel expenses. Payments made to individuals must not be so large as to induce individuals to risk harm beyond that which they would usually undertake.*

5. No information about the evaluation or materials was intentionally withheld from the participants.

*Withholding information from participants or misleading them is unacceptable without justifiable reasons for doing so. Any projects requiring deception (for example, only telling participants of the true purpose of the study afterwards so as not to influence their behaviour) are deemed high risk and require approval from the relevant C-REC.*

6. No participant was under the age of 18.

*Any studies involving children or young people are deemed to be high risk and require ethical approval through the relevant C-REC.*

7. No participant had a disability or impairment that may have limited their understanding or communication or capacity to consent.

*Projects involving participants with disabilities are deemed to be high risk and require ethical approval from the relevant C-REC.*

8. Neither I nor my supervisor are in a position of authority or influence over any of the participants.

*A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any study.*

9. All participants were informed that they could withdraw at any time.

*All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script (see Appendix 1).*

10. All participants have been informed of my contact details, and the contact details of my supervisor.

*All participants must be able to contact the investigator and/or the supervisor after the investigation. They should be given contact details for both student and supervisor as part of the debriefing.*



11. The evaluation was described in detail with all of the participants at the beginning of the session, and participants were fully debriefed at the end of the session. All participants were given the opportunity to ask questions at both the beginning and end of the session.

*Participants must be provided with sufficient information prior to starting the session, and in the debriefing, to enable them to understand the nature of the investigation.*

12. All the data collected from the participants is stored securely, and in an anonymous form.

*All participant data (hard-copy and soft-copy) should be stored securely (i.e. locked filing cabinets for hard copy, password protected computer for electronic data), and in an anonymised form.*

**Project title:** Attendance Monitoring Visualisation

**Student's Name:** Daniel Bates

**Student's Registration Number:** 22004544

**Student's Signature:** 

**Date:** 23/04/2023

**Supervisor's Name:** Dr Paul Newbury

**Supervisor's Signature:** 

**Date:** 25.04.2023

# **User Testing Compliance Form for UG and PGT Projects\***

## **Appendix 1: Introduction and Debriefing Scripts**

[This Study used a Google Form that contained this script]

### **Introduction Script**

#### **Attendance Monitoring Visualisation Final Year Project, 2022-23 Daniel Bates**

The aim of this study is to investigate the usability of the user interface for an Attendance Monitoring system. In order to come to a conclusion on how usable this interface is without biases, we need to ask for user feedback. You will have some time to interact with the website while completing 10 tasks and are asked to fill in a rating form from 1 to 10 for how usable the interface was each task.

I will be observing you while you perform the tasks and your form responses will be recorded as results for analysis. If you have any questions please ask me and let me know when you have finished or wish to withdraw.

I may ask you some general questions at the end of the study. Please remember that it is the system, not you, that is being evaluated – you are not being tested on your ability to perform the prescribed tasks.

You are welcome to withdraw from the study at any time. Please be assured that all data collected will be stored securely and the results anonymised for reporting.

If you agree to take part in this evaluation please select 'I agree'. If you have any questions before you start, please tell me before you progress to the next section.

[User is asked: "Please attempt each task and rate how usable the application was for this task from 1-10" and told "There will be an opportunity to provide textual feedback at the bottom of this page."]

### **Debriefing Script**

#### **Attendance Monitoring Visualisation Final Year Project, 2022-23 Daniel Bates**

The main aim of the experiment was to investigate the usability of this user interface. In particular, I was looking to see how clear/confusing the navigation bar, and user interface elements like selection boxes were, and if there were any traps that users may fall into.

Do you have any comments or questions about the experiment?

[Long text box for user to fill in optionally]

[On the post-submission 'Your response has been recorded' page]

Thank you for your help. If you have any further questions please contact myself at 'db524@sussex.ac.uk', or my supervisor at 'p.newbury@sussex.ac.uk'.

## **Appendix D   Project Proposal**

Can be found on the next page...

# AMVis: A web-based Attendance Monitoring Visualisation solution

Student: Daniel Bates

Candidate Number: 234558

Supervisor: Dr Paul Newbury

## Aims

Attendance and attainment in higher education have a proven link, and there is a large demand of higher education in the modern world. These result in a mammoth administrative task of monitoring the attendance for large cohorts of students. This is a problem that is only growing for the University of Sussex and similar institutions as they take on more and more students. There is no current application that fits the niche of visualising attendance data for this purpose. Therefore, there is considerable need for an application that takes in raw attendance data and creates meaningful and actionable visualisations for use by administrative staff, as well as configurable filters for these staff to extract visualisations of data they select.

This project will therefore aim to begin filling this niche with meaningful visualisations, stemming from user feedback, and aim to develop a configurable system for making such visualisations.

## Objectives

### Primary Objectives

1. Import attendance data into an effective and efficient database
2. Construct a well-structured and efficient database to store attendance data, and import sample attendance data
3. Create meaningful visualisations of attendance statistics for use by relevant staff
  - a. Per Student
  - b. Per Degree
  - c. Per Stage
  - d. Per Department & School
4. Get user feedback from relevant staff: in this case our Director of Student Experience, Dr Kate Howland, who due to her background in Interaction Design and as a user may be an asset in terms of specific feedback.
5. Apply this user feedback to improve visualisations and the system at large
6. Allow users to apply filters to data and create visualisations with this dataset

### Extensions

1. Create a full-stack web application around the visualisations, featuring
  - a. A ReactJS front-end
  - b. A Python (Flask) back-end
  - c. MySQL Database
2. Make the application user configurable, including different dashboards of visualisations
3. Investigate further ways to visualise attendance data e.g. Nightingale Rose Charts
4. Find meaningful statistics for student engagement beyond just quantity

## Relevance

This project sets out to research and deliver useful attendance visualisations to aid staff (primarily at the University of Sussex, but this could be extended beyond) concerned with attendance. As it is based around a web application, database, and data analysis, this project closely relates to my degree (Computer Science Integrated Masters) and has potential to inform decisions for my masters year, as well as my preferred career in Software Engineering, as that

much like my project, would include plenty of project and time management opportunities, as well as design around users which is currently being informed by the Human-Computer Interaction module.

This project will allow me to test and improve my skills in Databases following the module in second year, revise and extend my statistics knowledge from Mathematical Concepts, and investigate the domain of Data Science.

### Resources Required

This project will require use of my University of Sussex User Webpace for development though it may end up hosted elsewhere later. It will require use of School of Engineering and Informatics Lab machines for development where necessary.

### Weekly Timetable

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
0900	Lecture	Project / Coursework	Project / Coursework	Starting Lab Early	Lecture	Project / Coursework
1000				Lab Class		
1100				Lecture	Lab Class	
1200	Project / Coursework	Lecture				
1300		Lecture				
1400						
1500	Project / Coursework				Project / Coursework	
1600						
1700						
1800						
1900				Seminar		

### Rough Project Timescale

Late October/Early November: Plan and create a Database, importing the sample data.

November/December: Start producing visualisations and system backend. If any are good enough to show, then show to stakeholders and ask for feedback

December/January post-exams: Resume producing visualisations and system backend

February onward: Increase development speed with the increased module weighting, finish up core functionality and put development hours into most useful extensions

### Bibliography

*Database Handling in Python* (no date) *Code Envato Tuts+*. Available at:

<https://code.tutsplus.com/tutorials/database-handling-in-python--cms-25645> (Accessed: 10 October 2022).

Onojakpor, O. (2022) *A Python developer's guide to React, LogRocket Blog*. Available at:

<https://blog.logrocket.com/python-developers-guide-react/> (Accessed: 14 October 2022).

*React* (no date). Available at: <https://www.fullstackpython.com/react.html> (Accessed: 11 October 2022).

*The Data Visualisation Catalogue* (no date). Available at: <https://datavizcatalogue.com/index.html> (Accessed: 11 October 2022).

Kelly, G.E. (2012) 'Lecture attendance rates at university and related factors', *Journal of Further and Higher Education*, 36(1), pp. 17–40. Available at: <https://doi.org/10.1080/0309877X.2011.596196>.

### Interim Log

30/09: Initial Group Supervisor Meeting

03/10-06/10: Planning ahead of One-on-One Meeting – See Bibliography

07/10: One-on-One Supervisor Meeting

10/10-14/10: Writing and sending my Project Proposal