

Covid-Secure Computer Science Building – Solution Document

Group G

Ciaran Finnegan
Matthew Elliot

(This page does not count in the total, and is not marked)

MAX: 20 pages excluding this page, reference list and appendices

Introduction

In this document we give an overview for our implemented software solution for a Covid-Secure Computer Science Building. We explore its place in the market and future goals for the project. Ambitious planning was carried out before this project began and while many goals were achieved, others were not. We reflect upon the planning and development process to understand what was done right, what could be improved, and how to proceed into the future.

Background

The Covid-19 pandemic has drastically altered the way the world has operated in the past number of months. Government Guidelines & Legislation have forced nearly all public or communal spaces to be closed down, as we all know. For Universities, and the general workplace context, this has brought about the current “Work-From-Home” paradigm-shift, as-well as furlough schemes. These are working solutions for the height of the pandemic; however, they are only temporary. The Department for Education expects Higher Education providers to be reopened in 2021 [1]. However, this will come with risks. University students will be coming to the campus from many regions in the UK and abroad, bringing potential infections with them. Reopening university premises will require measures and systems to be put in place that prevent further outbreaks and closures. Furthermore, these measures should help students and university staff feel safe to return, otherwise reopened spaces will remain empty, causing a waste of resources and real estate.

To date, most efforts to ensure reopened environments are Covid-Safe have been physical, such as: requiring face coverings, providing sanitation-stations, restricting the number of occupants, physical barriers, etc. These measures are effective and essential in controlling the spread of infections, however, there is great potential in employing additional digital solutions, and Digital Transformations could play a key role in these solutions. Advice issued to students and employees has been to “act responsibly” and “take a sensible approach” to travel and study [2] [3]. In order for individuals to act responsibly it would help if they were informed and could make informed decisions. Of course, this would require data and information to be collected and made available to them. The collection of such data in work environments has been tremendously aided by Digital Transformations in recent years, as highlighted by CoWorkr & Haworth AP. In their report [4] they explain how building sensors have increased the resolution and amount of data being collected, compared to previous manual collection methods. They also illustrate the

decreasing price of sensors, which increases viability of this digital information collection system. One of the types of data highlighted in their report is location data. Using indoor sensors, it is possible track numbers of occupants and space usage throughout a day. This will be one of the key concepts utilised in our implemented system.

System Goals

The system we have developed has these key goals:

- Keep staff and students informed
- Help staff and students to act responsibly
- Increase safety of staff and students during the Covid-19 pandemic and beyond
- Enable a smoother transition between closure and reopening

To achieve these goals 3 main system components have been developed:

- OccupancyTracker: A Web-App that tracks and displays the real-time occupancy data for buildings, labs, and other spaces.
- Covid Self-Report Portal: A web portal that allows students to responsibly and anonymously submit a self-report for their Covid Diagnosis.
- Covid-Compliant Entry System: A building entry system that ensures anyone entering the building is following guidelines by wearing a face covering and denies access infected individuals.

The OccupancyTracker system will allow individuals to make informed decisions and act responsibly. They can decide if they think it would be safe and responsible to enter a building or space based on current guidelines and how occupied it is.

The Covid Self-Reporting portal will allow students to be responsible and inform university faculty about their Covid diagnosis, which will allow the university to keep track of infections and take proper precautions.

The Covid-Compliant Entry System will provide a first line of defence to ensure occupants remain safe, follow guidelines, and prevent high risks of infections in the building.

If the system is successful in achieving its goals, further closures and infection outbreaks will be kept to a minimum, and the university will maintain a balance of social distancing and face-to-face on-campus activity.

The system is not intended to be used in isolation to achieve the goal of a Covid-Safe environment, but we believe it can play an instrumental role, in conjunction with the

various physical measures that are currently put in place, in helping everyone on campus feel safer.

Scoping and market evaluation

There are a number of products on the market that solve similar problems as the ones we focus on. In the Software Design Solution, we had the idea of a Track & Trace system, however we decided against developing this for the MVP as the government Track & Trace systems in use are already quite effective and widely used. There are also other products marketed as real-time Occupancy Trackers for buildings and workplaces, however the majority of them all are focused providing insights to building owners and employers, such as the solutions from Axper [5] and PFM Footfall Intelligence [6]. Our system distinguishes itself by focusing on providing insights to building users, as well as owners. Other system on the market also tend to require bespoke proprietary sensors to gather information, such as Hoxton AI [7]. Our system allows for any web enabled sensor to connect to it via the Web API.

The thought of using sensors to detect if a person is compliant with coronavirus regulations is not new. Amazon's Panorama box lets firms check if staff follow coronavirus rules by monitoring if they are socially distancing and wearing facemasks in the workplace [8]. France is using this same technology to ensure facemask compliance on public transport [9]. Our entrance system takes this idea and uses it as one factor as to decide if a person would be allowed to enter the building. While basic machine learning models can be built and ran locally, many companies offer a machine learning service that abstracts the computationally expensive process of creating the model and then classifying test data against it. All of the large cloud computing vendors have this service to offer: AWS offers SageMaker [10], Google Cloud offers Vision AI [11] and Microsoft Azure offers CustomVision [12]. All have similar architectures and offerings in terms of using the service with training data to construct a custom model fit for a specific purpose and an endpoint to test images against the model.

Project Planning

Before development began, a team meeting took place where we discussed what features we believed would be feasible for our team in the given time-frame. From the ideas presented in the Software Design Solution, it was determined that the OccupancyTracker, Self-Reporting system and Covid-Compliant entry system would make up the Minimum Viable Product, or MVP, and the system could be extended with further features if time allowed. From here we decided on a list of main User Stories for each feature. We used Monday.com as a project management tool to keep track of user stories and feature progress using Kanban boards. Our Kanban board would have 4 sections: Backlog, Developing, Testing, Release. Given the small team size, we decided that each team member would work on developing different features alone so that we could focus on completing features while evenly distributing the workload. Originally, we had planned to use a Scrum workflow, with weekly sprints. At the beginning of each week we would discuss what user stories we aim to complete. Short daily online “stand-ups” would take place where we discuss current progress, and then at the end of the week we would hold a retrospective to discuss how the sprint went, what we achieved, what could be improved, and how to progress. The GitHub repos for features were shared with each other so that we could see each other progress. We planned to communicate daily over Microsoft Teams, as it has useful teams and meeting features.

Unfortunately, the team wasn’t able to fully realise and carry out our envisioned project process. Due to time constraints, small team size, other engagements we had to opt for a more informal ad-hoc Agile approach. Most of the development progress was made in the final 1-2 weeks of the timeline, and user stories were completed as needed without formal planning meetings or tests as we believed they could take up valuable development time needed to reach the MVP. Frequent informal progress updates were exchanged to ensure each team member was on track and help was offered when needed.

From this we have learned that after conducting planning at the beginning of the project, progress should begin immediately at a slower consistent pace allowing for proper testing, further planning and discussion. Future development extending the MVP will certainly aim to follow this more structured development and planning process.

Software Realisation

OccupancyTracker

The OccupancyTracker is a web application that uses information received from sensor devices to keep track of the occupancy of various buildings and spaces in the university. In our original concept, only the Computer Science Building was considered, but it was found that expanding the system to work for many buildings across campus was quite viable, the main limitation is the actual building's infrastructure being able to accommodate the sensors needed to receive and send information.

The application has a frontend website with main 3 pages. A Buildings pages, Labs page, and Spaces page. The Buildings page displays the list of buildings on the database, with their occupancy rate. The user can click on a building to see the list of Spaces inside that building. and their occupied status. The Spaces page shows the list of all Spaces in the database, and if they are occupied or not. The Labs page shows the list of computer labs and a percentage for how occupied they are.

The system also provides a RESTful Web API as an interface for communication from building sensors.

System Design

The backend of the application relied upon a database, which holds the information on buildings, spaces, computers, labs, and their occupancy. *Appendix 2 - Figure 4* shows the schema of this database.

The application code utilised a standard ASP.NET Core Model-View-Controller design pattern. This MVC pattern allowed us to decouple the user interface, data model and controllers, which allowed us to focus on one aspect of the application at a time. To communicate with the database, we used the .NET tool Entity Framework Core. EF Core is an object relational mapper, a powerful tool that enables us to work with databases easily using C# objects. The controllers handle requests from the web, get information from the database via a Database-Service layer utilising EF Core, and return the information to the frontend in a View for the user. Separate API Controllers are used to handle API requests from sensors and update the database with the information provided in the request.

S.O.L.I.D design principles [13] were used for this application to improve structural clarity and maintainability. Following the single responsibility principle, the application was split into 3 layers. The data layer, database service layer, and the application layer. The data layer contains the entity Models and the Database context which would represent the

database and its entities using .NET objects. The database services layer contained separate services for each entity in the database, which communicated with the database via EF Core. Each service inherited from an interface, allowing for dependency injection to be used as per the Dependency Inversion principle. Using dependency injection and abstractions would mean implementations of the services could be easily swapped out without needing to go through and update every use of the service, in addition to making future testing simpler.

System Requirements:

Requirement	Description
Availability	The system should be lightweight and be able to handle a peak concurrent use time of 10am – 7pm, Monday – Friday, with a maximum of 5% downtime.
Accuracy	The system should accurately track occupancy information in real-time, with a maximum 10% error.
Usability	The frontend of the application should be intuitive, easy to use, and informative.
Communication	The system must provide an interface by which sensors, devices, and admins can send it information.
Scalability	The application should be able to handle a peak load of 60% of the University student population during the exam period, and 50% during the semester.

To meet the availability & scalability requirements, the application was packaged into a lightweight docker container and uploaded to Azure container services for hosting. The database used was also located on an Azure SQL Server. Using cloud container services allowing for elastic scaling to current usage needs with the likes of Kubernetes. Cloud Database servers also make backups scaling much easier.

To maintain accuracy of the system, sensors should be placed in all entrances of the buildings to ensure all entrances and exits are recorded. A full list of computers in the labs should also be kept in the database so that lab occupancy is accurate.

Bootstrap, a popular HTML styling library, was used for the frontend of the site. This helped create an attractive and professional looking UI that should be intuitive for all users.

To enable communication between the application and the sensors, the RESTful Web API was created. Nearly all web enabled devices are capable of send HTTP requests to an endpoint, therefore this ensures the application can be fully decoupled from the sensors and can receive the data from a wide range of IoT devices.

Logical View

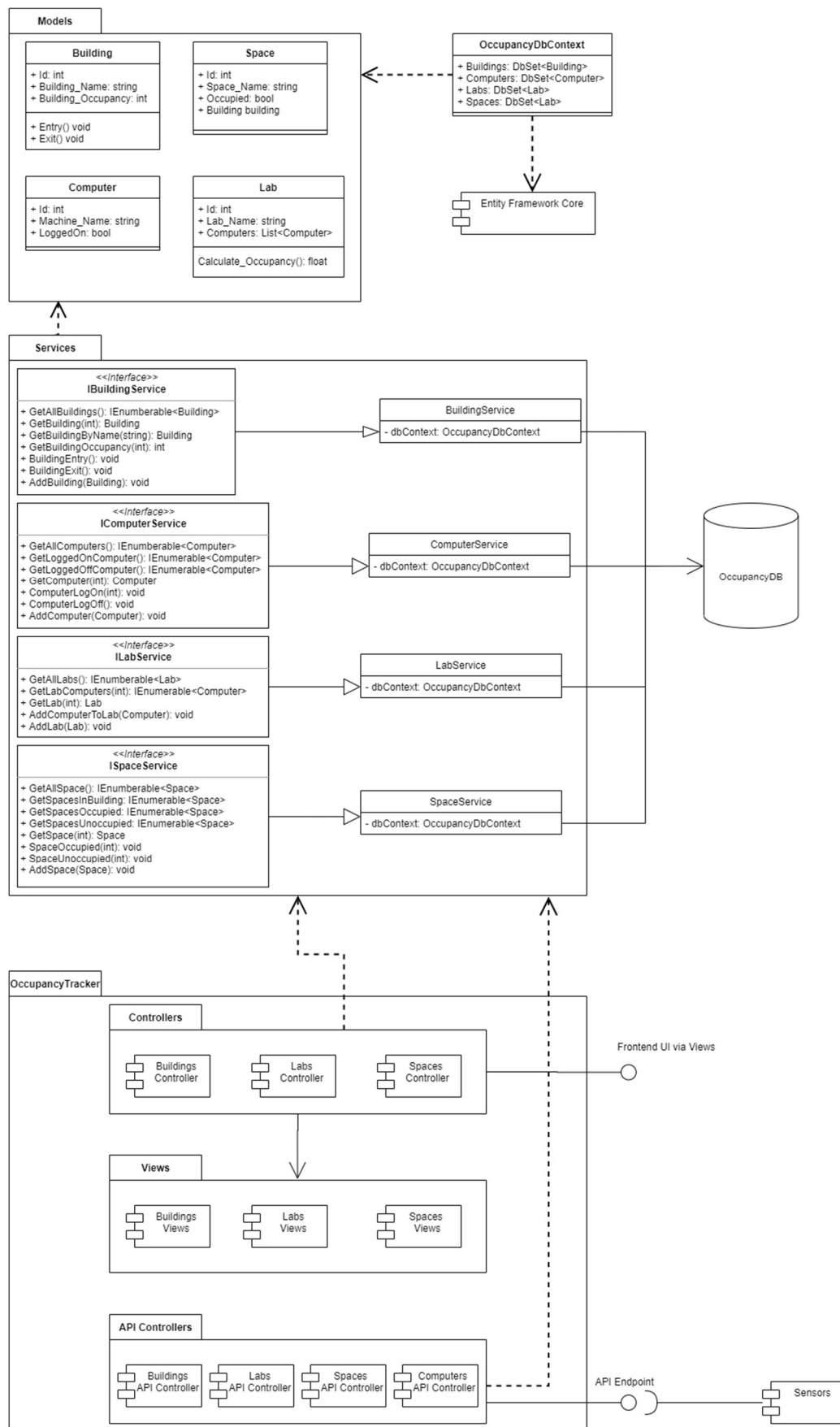


Figure 1: OccupancyTracker Logical View

Covid Self-Report System

The Covid Self-Report System is a simple web application with 2 pages, The report page, and successful submission page. The report page contains a form for users to submit their student number, diagnosis date and contact number. The information submitted by the form is then added to the Report database.

System Design

The database is also very simple and contains one table. *Appendix 2 - Figure 5* shows the database schema.

This application had the same development and design principles as the OccupancyTracker system but applied to a two page design. It was an ASP.NET Core MVC Web Application, using Entity Framework Core as the Object Relational Mapper.

System Requirements

Requirement	Description
Availability	The Self-Report system should maintain 100% availability for the foreseeable future of the pandemic
Confidentiality	The system should not reveal or collect unnecessary personal information on infected students
Security	Reports should only be accessed by authorised QUB staff
Usability	The application should work on all modern browsers: Google Chrome, Safari, Firefox, Edge.

Much like the OccupancyTracker, this system was packaged into a Docker container and hosted on Azure container services. As this program was already quite lightweight, utilising cloud based container hosting would ensure the application availability wouldn't be an issue. Although, it is unlikely that the system will have a large number of concurrent users attempting to submit reports at the same time, the system should be able to handle should an event.

To keep confidentiality for student's identity, the application uses a minimum amount of information about the students. Only the student number, phone number, and diagnosis date are requested, and any further information would be unnecessary. The web page also displays a note that the information they submit may be used to warn students who they may have come into contact with, but that their personal information will never be disclosed. At the moment this notification feature has not been implemented, but the message makes sure users are aware incase the feature is implemented in the future.

The backend database is also hosted on an Azure SQL Server. Accessing the database requires login credentials. Additionally, the server can have different roles and security policies for server logins, controlling what users can see what information. The ReportDB will be accessible only by authorised QUB staff who can take precautionary measures based on reports.

Again, Bootstrap was used for this application's frontend UI to provide a modern looking intuitive interface. The report submission contains only a straightforward web form to avoid user confusion.

Logical View

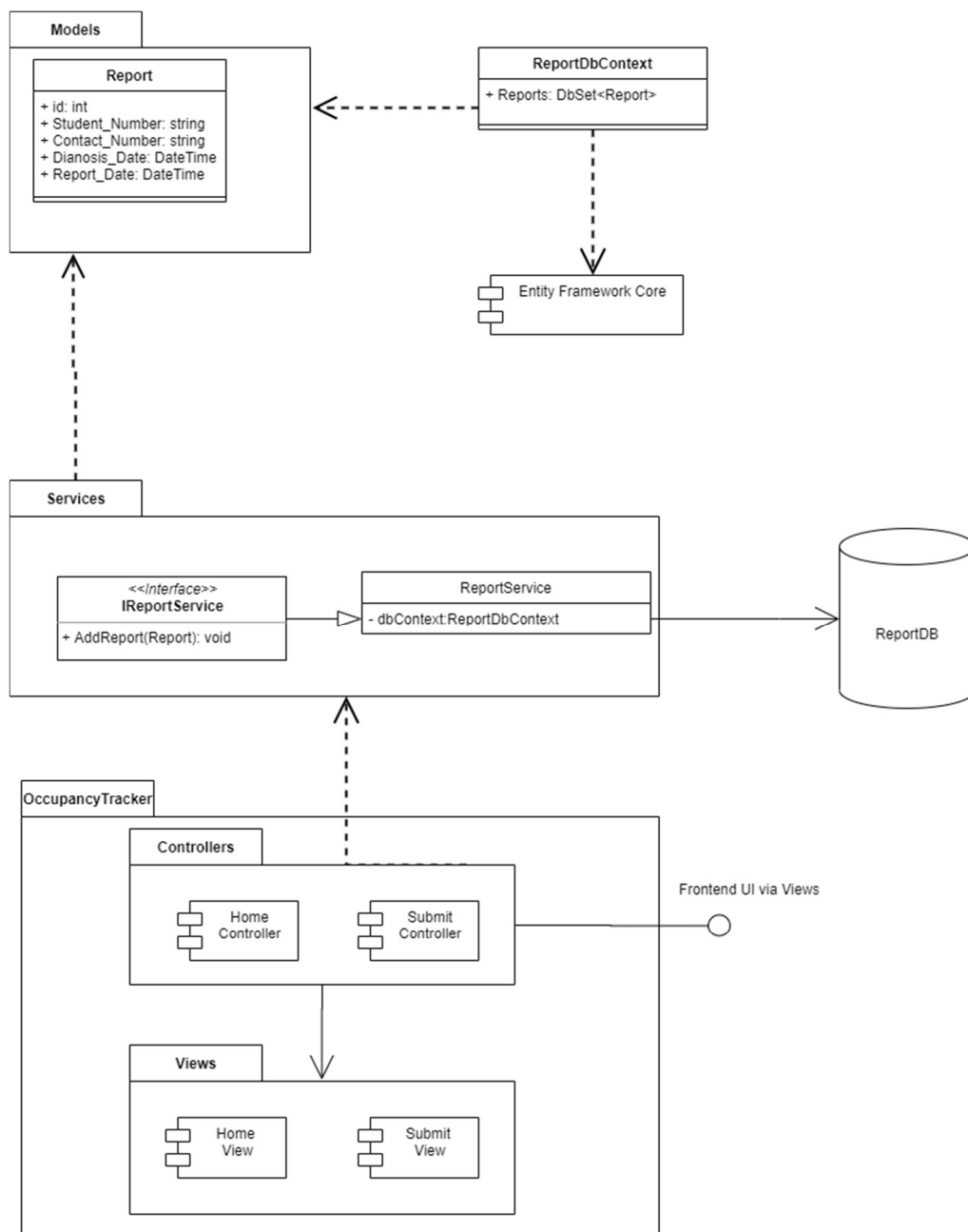


Figure 2: Self Report Logical View

Entrance System

An entrance has the opportunity to become the first line of defence against Covid by enforcement of regulations such as quarantine and facemask wearing to keep everyone safe and the building open.

System Requirements

Here is a table of the summarised requirements for the entrance system.

Requirement	Description
Availability	The system should maintain 100% availability of at least one entrance system per space.
Throughput	1 Person per second through the CSB main entrance
Scalability	An unlimited number of entrance systems should be able to work in tandem.
Usability	Easy and intuitive for a student to operate. Will allow for exemptions to certain rules where required.
Performance	The camera will adjust to the height of the person to ensure the image is taken at the correct angle in less than 0.3 seconds. The images will be processed, and the results calculated in less than 0.4 seconds.
Security	Allow entry for only those who have a valid student number. Ensure that no external traffic can access the running server.

As seen from above, there were many requirements set forward to be achieved. One requirement was less than a 1% true negative rate. This was surpassed with a 0% true positive and 100% overall accuracy from the trained ML model. *See Appendix 6, Figure 15.*

The throughput of 1 person per second into the CSB coming from two entrance systems that could let in one person every 2 seconds was required. The actual average time of the decision being made takes 2.7 seconds which resulted in it taking 1.35 seconds per admission, assuming the 2 entrance systems were implemented.

Another requirement that was fulfilled was the ability for the system to not discriminate against students that have an exemption to wearing a mask. The system will first check that they do not have a mask exemption before checking if they are wearing a mask.

The system is highly modifiable. The checks that the system preforms are boolean and come as a sequential list. Adding or removing different checks that the system preforms is extremely easy.

System Design

The entrance system can be divided into four main parts: (a) a simple GUI that a student can interact with, (b) a database that can store the necessary information on the student, (c) a machine learning model that can determine if a student is wearing a mask, (b) a flask web server for running the entrance service, responsible for displaying the GIU and managing backend tasks

The GUI is responsible for displaying a live video feed to the student beside an NFC card reader. The GUI is built using HTML and JavaScript. The WebcamJS library is used to display the live feed from the camera to the student. JavaScript is used to link the available button and text box (for manual student card entry, used in this prototype) to the flask server that is running in the background in order for the server to react to the actions of the student.

Azure SQL databases are used to keep track of who has recently self-reported as testing positive for Covid as well as information from a potential contract tracing system based on personal CSB usage information. Not everyone can wear a facemask for medical reasons. As to not discriminate, a database of who is exempt from wearing a facemask also exists. Both of these databases are centrally stored on an Azure database server and accessible by any individual entrance system.

In order to determine if a student is wearing a facemask, an Azure CustomVision model was trained with a dataset of 644 faces with a mask and 635 faces with no mask. The person in each image varied on skin colour, hair colour, gender, lighting, quality, and background in order to make the model more robust under real-world conditions. The model can be used to determine whether a student is wearing a mask. An endpoint is provided which can be used to access this trained model and return a response as to the probability of the person wearing and not wearing a mask.

The flask server acts as the backbone that joins the above three parts together. When a student card is entered via the NFC reader or manual input (for the prototype), the server starts a series of sequential checks beginning with the quickest and least expensive tasks in order to reduce both the time taken for the system to come to a conclusion. the system will check that they have not been in close contact with anyone else that has texted positive from. This data should come from the contact tracing system. Next, the system will check that they are not a positive case them self. This data should come from the self-reporting system. If these checks pass, the system will then check if they are exempt from wearing a face covering. If they are, the door will open, and they will be granted entry. Otherwise, the

camera will take a picture of their face, send it to the CustomVision machine learning model and open the doors if they are wearing a mask. If they are not, they will be asked to put one on and try again.

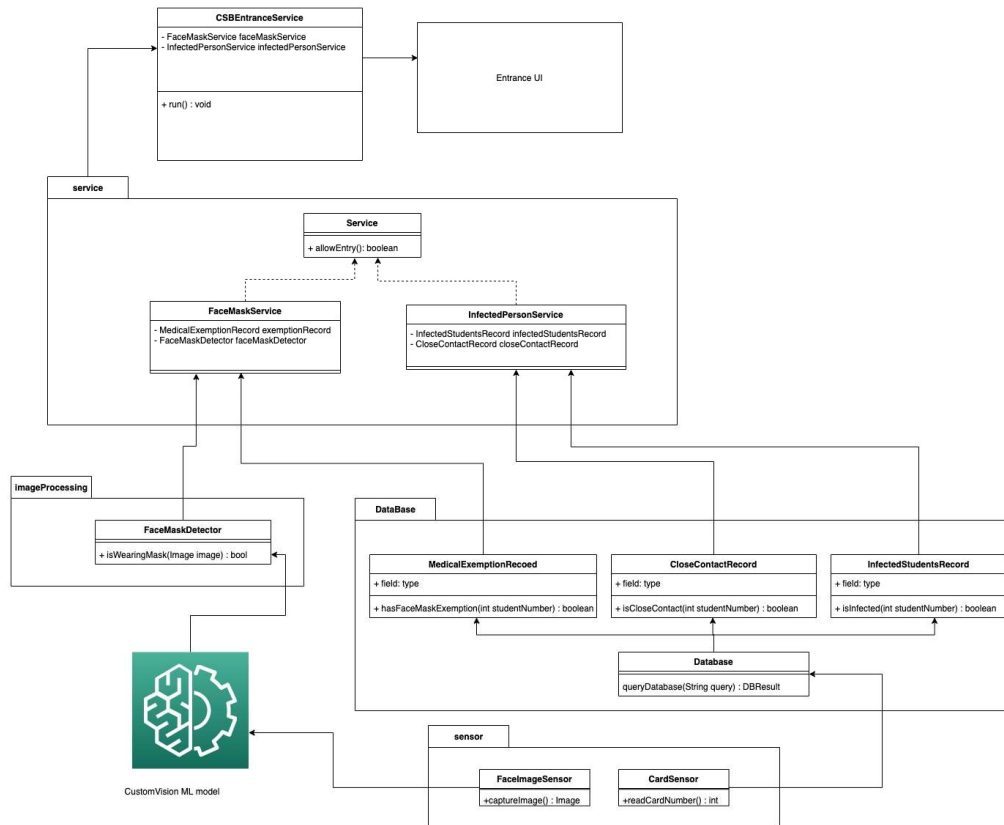


Figure 3 Entrance System Logical View

The python flask server was used for three reasons. Firstly, it is a lightweight web server capable of running on relatively low power devices. Secondly, it is very simple to set up, giving allowing for swift implementation. Finally, python allows for easily calling on the Azure services to get information on when is an infected contact, who has a mask exemption as well as preforming complicated tasks such as determining if an image of the student's face is wearing a face mask.

Documentation

OccupancyTracker

Home Page

When you first load the application webpage you are greeted by the Home page. This contains a carousel of images. This could be used to show Covid updates, but for now displays images of the campus and a brief description of the purpose of the application.

The top of the page holds a standard navigation bar. The navigation bar has tabs for all the pages on the site: Buildings, Labs, Spaces, as well as the Home page. See *Appendix 3, Figure 6*.

Buildings

Navigating to the Buildings Page displays a list of all the buildings on the system. Each building shows its current number of occupants on the right.

Clicking on one of the buildings in this list will bring the user to that building's details page. Here we can also see the building occupancy, as well as all the spaces in that building. Beside each Space is its occupancy status. See *Appendix 3, Figure 7 & 8*.

Labs

Navigating to the Labs page displays a list of all the Labs in the system. Each lab shows how full they are, displayed as a percentage bar for quick identification of capacity. This is calculated by dividing the number of logged on computers in the lab by the total number of computers in the lab. See *Appendix 3, Figure 9*.

Spaces

The final page on the site is the Spaces page. Navigating to this page displays a quick overview of the occupancy status of all the spaces on the system, and what building they are in. See *Appendix 3, Figure 10*.

API

To communicate with sensors in the buildings, the application exposes a RESTful Web API that handles HTTP requests. The API can be used to update information in the database, as the sensors would do. The API also allows information to be retrieved from the database, or even added to the database. This will be useful for administrators to check the status and test the database without having to connect to a server management tools. *Appendix 7* shows the list of API endpoints available. All endpoints are preceded by `http://[url]/api/`

Covid Self-Reporting Portal

Navigating to the Covid Self-Reporting site brings the user straight to the report submission page. The page contains a simple form where the user can submit their student number, contact number, and diagnosis date. The student number must contain between 7 and 8 numerical characters to be valid. Upon a successful report submission, the user is brought to the success page. The success page displays a message to let the user know that their report has been submitted and encourages them to self-isolate. See *Appendix 4, Figure 11 & 12*.

Entrance system

The entrance system only requires a single page for the GUI. You will see a live video feed of what the systems camera is seeing. Below that, there is an option to manually enter your student number or you can simply scan your student card at the NFC contact point beside the display and the entrance system will automatically start to determine if you should be admitted. If you are denied entry, a reason will be shown to the user. See *Appendix 5, Figure 13 & 14*.

When setting up the system, a readme document is provided. It will tell you to obtain the “*config.json*” file with all the confidential information to access Azure. It will tell you to run the command “python3 App.py” and navigate to “http://127.0.0.1:5000/”. That is all that is required to set up.

Legal, social and Ethical Implications

What happens to your personal data is very important. If leaked into the wrong hands, a lot of damage to both the student and the reputation of QUB will be affected. Careful consideration was given as to what data was stored and how it was used. Organisations have a legal responsibility to ensure data they hold on individuals is being handled and stored reasonably according new GDPR laws and the Data Protection Act.

In the entrance system and Covid self-report system, only the student’s number was used to identify them with being infected, being a close contact or having a medical exemption. No system logs are kept of these queries to the remote databases storing this information. Furthermore, when the system takes a picture of the students face to send it to be analysed by the ML model, the image is not permanently stored in the local system. It will be overwritten when the next person used the system. On the Covid self-report system, a note tells the user that the information may be used to warn other users. This

ensures that users are aware of how the information they submit is used. The report system requires the user to offer very little information that the university doesn't already have, as the university already has access all student numbers, and their contact preferences. The reporting system would be completely optional, meaning students have no obligation to inform the university of their health status, but opens the option for students who wish to.

Microsoft Azure SQL Server databases and CustomVision machine learning models were used and thus the implications considered. Special consideration of sending a picture of a student's face to the Azure CustomVision machine learning model was considered. Microsoft Azure operated under the ISO/IEC 27018 Code of Practice [14] for Protecting Personal Data in the Cloud. This means we will know where our data is stored and what's happening with our personal identifiable information. Furthermore, our data will not be used for marketing or advertising without explicit consent (which will not be given). Microsoft complies only with legally binding requests for disclosure of customer data

Critical Analysis and Lessons Learnt

The goal of any smart building transformation is to increase the efficacy and decrease the burden of management for the owners while improving the experience of the occupants in the building. While some ideas may sound like good ideas on paper, it is easy to implement a costly system that increases the burden of upkeep of the building and reduces the experience of the occupants by putting in place unnecessary steps.

One such example of this is the entrance system. The burden of maintenance could be increased when systems do not work as planned. What if there is a problem with the data coming from the contact tracing system that the entrance system relies on? What about if there is an emergency where police or the fire-service need quick entrance to the building? What if there are poor lighting conditions that means the system cannot check if you are wearing a mask? Careful consideration and sandboxing of ideas need to take place to map out and plan for unexpected events.

As mentioned in the previous section, data security is very important for modern digital solutions. While the occupancy tracker does not hold any important personally identifiable information, it does expose a public API that can be used by anyone. The original intention was to employ Token Authentication / Authorisation for the API so that only authorised users and machines could use the API, however implementing this into the system proved more troublesome than expected. Therefore, the security of the system in the current state is not as adequate as we had hoped.

Potential Business Context

The systems in this project have potential to form marketable business products with some more development. Many businesses, stores, and public spaces are enforcing strict mask policies in their premises. Mask policy enforcement is currently typically done manually by employees near the entrance of the premises. An automated entry system based on computer vision, such as our Covid-Compliant entry system, would allow staff to carry out other important duties for the premises, saving time, resources, and therefore money.

Given a time frame of roughly 1 month of continued development on the system, it could be elevated to a place suitable to present to potential customers before the pandemic situation comes to an end, restoring things to normal and eliminating the need for this product.

The OccupancyTracker system however could be a viable business product for Covid and beyond. University campuses have many buildings and spaces, with thousands of students. It is sometimes difficult to find space to study on campus, especially during exam periods. Many students have experienced frustration after travelling to their campus or library, only to find there were no spaces available for them. The OccupancyTracker system would allow students to check occupancy to assess if it would worth while travelling to campus. The system can apply to premises other than universities. Public libraries, car parks, and other communal areas may make use of such a feature. Continued developed over a time frame of 2 months would allow the team to further develop the product to state that will be suitable for potential clients after the Christmas holidays when lockdown restrictions should be less strict allowing for less closures, while users still would like peace of mind about the safety of the space.

Product Roadmap & Future Planning

Going into the future, if development is continued, our main concern would be acquiring more talent / team members. With only 2 people working development, progress is slow and proper testing had been pushed to the side in order to meet the MVP. Increasing the team size to a minimum of 4 would allow for adequate development and testing to be done. The best course of action would be to form a full team with backend and frontend developers, testers, and project managers if a budget would allow for this.

OccupancyTracker Roadmap

Continuing development on the OccupancyTracker as product would begin with retroactive formal testing of all the features in the system. This would find any bugs as early as possible, allowing them to be resolved as soon as possible. Automated Unit tests and Integration testing should then be written. A group of team members should work on employing a Continuous Integration / Continuous Deployment pipeline to streamline the release process, with successful automated tests triggering release builds after a repo push. Team members with design experience should work updating the UI to have more personality and a unique look to give the feel of a quality product.

Several new features would be desirable to include. Updated security measures would be essentials. Bearer token authentication will be implemented for the Web API restricting access to only authorised connections.

A new administrator page should be created on the frontend that will also entities to added and edited in the system by admins via the UI, this would make it more marketable to less technically savvy clients who wouldn't like/know to interact via the API or database.

These points should transform the system into a viable product within the projected timeline, while giving enough time for refinement and unexpected problems.

Entrance System Roadmap

Moving from the prototype to real world implementation, safety must be of the highest priority. A function to unlock all doors should be implemented. The entrance system should be connected to the fire alarm to allow it to unlock all doors at once.

Currently, it takes 2.7 seconds to perform all checks sequentially to allow the doors to open. This is too slow and will act as a nuisance. The checks should be moved to operate in parallel, theoretically decreasing to the time to allow entry to below one second.

A frontend that allows system administration to manage the entrance systems should be developed. At first, relatively simple features like temporally disabling an entrance system due to an unforeseen circumstance should be implemented. Thinking longer term, a control panel should be created that will let building managers visualise where the entrance systems are and what checks they are performing. After this is implemented, they should be able to dynamically control which entrance systems are active and what checks they are performing.

References

- [1] Department for Education, “Higher Education - Reopening buildings and campuses,” December 2020. [Online]. Available: <https://www.gov.uk/government/publications/higher-education-reopening-buildings-and-campuses/higher-education-reopening-buildings-and-campuses>. [Accessed 9 December 2020].
- [2] Department for Education, “Students returning to higher education for spring term,” December 2020. [Online]. Available: <https://www.gov.uk/government/publications/higher-education-reopening-buildings-and-campuses/students-returning-to-higher-education-from-spring-term>. [Accessed 09 December 2020].
- [3] Public Health Agency, “Coronavirus (Covid-19): Information for students,” October 2020. [Online]. Available: https://www.publichealth.hscni.net/sites/default/files/2020-10/COVID%2019%20Students%20Factsheet%20A4%2010_20.pdf. [Accessed 9 December 2020].
- [4] J. Aarmidor, “How Embedded Sensors will Transform Workplace Performance, Employee,” CoWorkr and Haworth AP, 2017.
- [5] “Real Time Occupancy and Social Distance Tracking,” Axper, [Online]. Available: <https://axper.com/real-time-building-occupancy>.
- [6] “Real-time occupancy management in retail during and after COVID-19,” PFM - Footfall Intelligence, [Online]. Available: <https://www.pfm-footfall.com/occupancy-management/>.
- [7] Hoxton AI, “Real Time Occupancy Alerts,” [Online]. Available: <https://www.hoxton.ai/home>.
- [8] J. Vincent, “France is using AI to check whether people are wearing masks on public transport,” The Verge, 7 May 2020. [Online]. Available: <https://www.theverge.com/2020/5/7/21250357/france-masks-public-transport-mandatory-ai-surveillance-camera-software>.
- [9] AWS, “Amazon SageMaker,” [Online]. Available: <https://aws.amazon.com/sagemaker/>.
- [10] Google, “Vision AI,” [Online]. Available: <https://cloud.google.com/vision>.
- [11] Microsoft, “Custom Vision,” Microsoft, [Online]. Available: <https://azure.microsoft.com/en-gb/services/cognitive-services/custom-vision-service/>.
- [12] S. Oloruntoba, “S.O.L.I.D: The First 5 Principles of Object Oriented Design,” Digital Ocean, 21 September 2020. [Online]. Available: https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design. [Accessed 09 December 2020].
- [13] ISO, “ISO/IEC 27018:2019,” [Online]. Available: <https://www.iso.org/standard/76559.html>.
- [14] BBC, “Amazon's Panorama box lets firms check if staff follow coronavirus rules,” December 2020. [Online]. Available: <https://www.bbc.com/news/technology-55158319>.

- [1] Department for Education, “Higher Education - Reopening buildings and campuses,” December 2020. [Online]. Available: <https://www.gov.uk/government/publications/higher-education-reopening-buildings-and-campuses/higher-education-reopening-buildings-and-campuses>. [Accessed 9 December 2020].
- [2] Department for Education, “Students returning to higher education for spring term,” December 2020. [Online]. Available: <https://www.gov.uk/government/publications/higher-education-reopening-buildings-and-campuses/students-returning-to-higher-education-from-spring-term>. [Accessed 09 December 2020].
- [3] Public Health Agency, “Coronavirus (Covid-19): Information for students,” October 2020. [Online]. Available: https://www.publichealth.hscni.net/sites/default/files/2020-10/COVID%2019%20Students%20Factsheet%20A4%2010_20.pdf. [Accessed 9 December 2020].
- [4] J. Aarmidor, “How Embedded Sensors will Transform Workplace Performance, Employee,” CoWorkr and Haworth AP, 2017.
- [5] “Real Time Occupancy and Social Distance Tracking,” Axper, [Online]. Available: <https://axper.com/real-time-building-occupancy>.
- [6] “Real-time occupancy management in retail during and after COVID-19,” PFM - Footfall Intelligence, [Online]. Available: <https://www.pfm-footfall.com/occupancy-management/>.
- [7] Hoxton AI, “Real Time Occupancy Alerts,” [Online]. Available: <https://www.hoxton.ai/home>.
- [8] BBC, “Amazon's Panorama box lets firms check if staff follow coronavirus rules,” 2 December 2020. [Online]. Available: <https://www.bbc.com/news/technology-55158319>.
- [9] J. Vincent, “France is using AI to check whether people are wearing masks on public transport,” The Verge, 7 May 2020. [Online]. Available: <https://www.theverge.com/2020/5/7/21250357/france-masks-public-transport-mandatory-ai-surveillance-camera-software>.

- [10] AWS, “Amazon SageMaker,” [Online]. Available: <https://aws.amazon.com/sagemaker/>.
- [11] Google, “Vision AI,” [Online]. Available: <https://cloud.google.com/vision>.
- [12] Microsoft, “Custom Vision,” Microsoft, [Online]. Available: <https://azure.microsoft.com/en-gb/services/cognitive-services/custom-vision-service/>.
- [13] S. Oloruntoba, “S.O.L.I.D: The First 5 Principles of Object Oriented Design,” Digital Ocean, 21 September 2020. [Online]. Available: https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design. [Accessed 09 December 2020].
- [14] ISO, “ISO/IEC 27018:2019,” [Online]. Available: <https://www.iso.org/standard/76559.html>.
- [15] BBC, “Amazon's Panorama box lets firms check if staff follow coronavirus rules,” December 2020. [Online]. Available: <https://www.bbc.com/news/technology-55158319>.

Appendices

Appendix 1 - Contributions

Table 1: Individual Contributions

Item	Contributions
OccupancyTracker System	Ciaran – 100% https://github.com/cfinnegan12/OccupancyTracker https://hub.docker.com/repository/docker/cfinnegan12/occupancy-tracker http://occupancy-tracker.westeurope.azurecontainer.io/
Covid-Compliant Entry System	Matthew – 100% https://github.com/matthew-64/CSBEntranceSystem
Covid Self-Reporting System	Ciaran – 100% https://github.com/cfinnegan12/CovidSelfReportingPortal https://hub.docker.com/repository/docker/cfinnegan12/self-report-portal http://covid-self-report.westeurope.azurecontainer.io/
Solution Document	Introduction – Ciaran 100% Background – Ciaran 100% Scope and Market Evaluation – Ciaran 50% , Matthew 50% Project Planning – Ciaran 100% Software Realisation – Ciaran 50%, Matthew 50% Documentation – Ciaran 50%, Matthew 50% Legal, Social, and Ethical Implications – Matthew 100% Critical Analysis & Lessons Learnt –Matthew 100% Potential Business Context – Ciaran 50%, Matthew 50% Future Planning & Roadmap – Ciaran 50%, Matthew 50%

Appendix 2 – Database Schemas

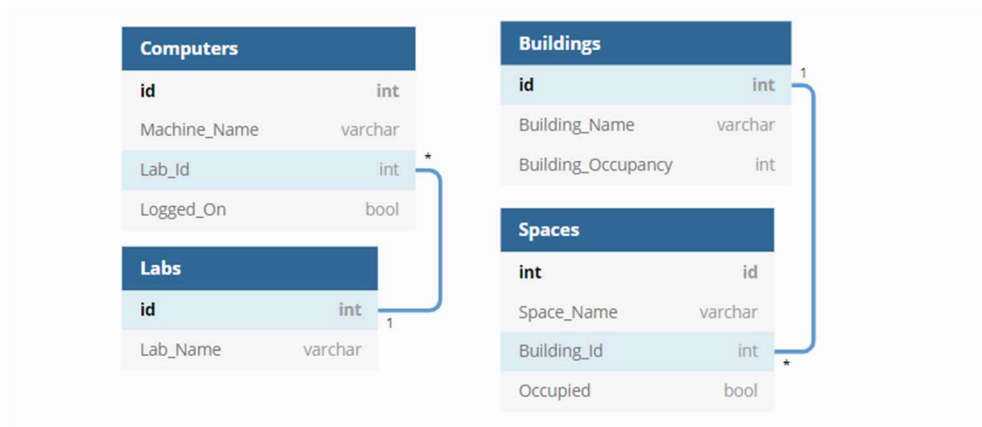


Figure 4: OccupancyDB Schema

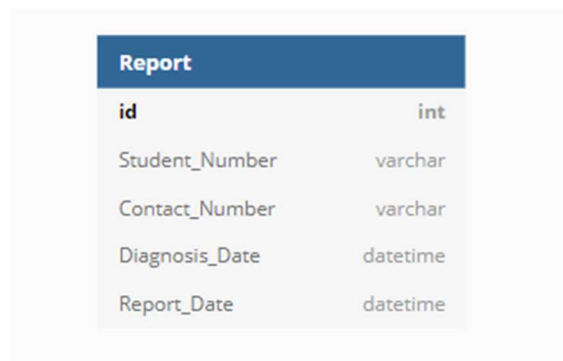


Figure 5: Report Database Schema

Appendix 3 – OccupancyTracker Screenshots

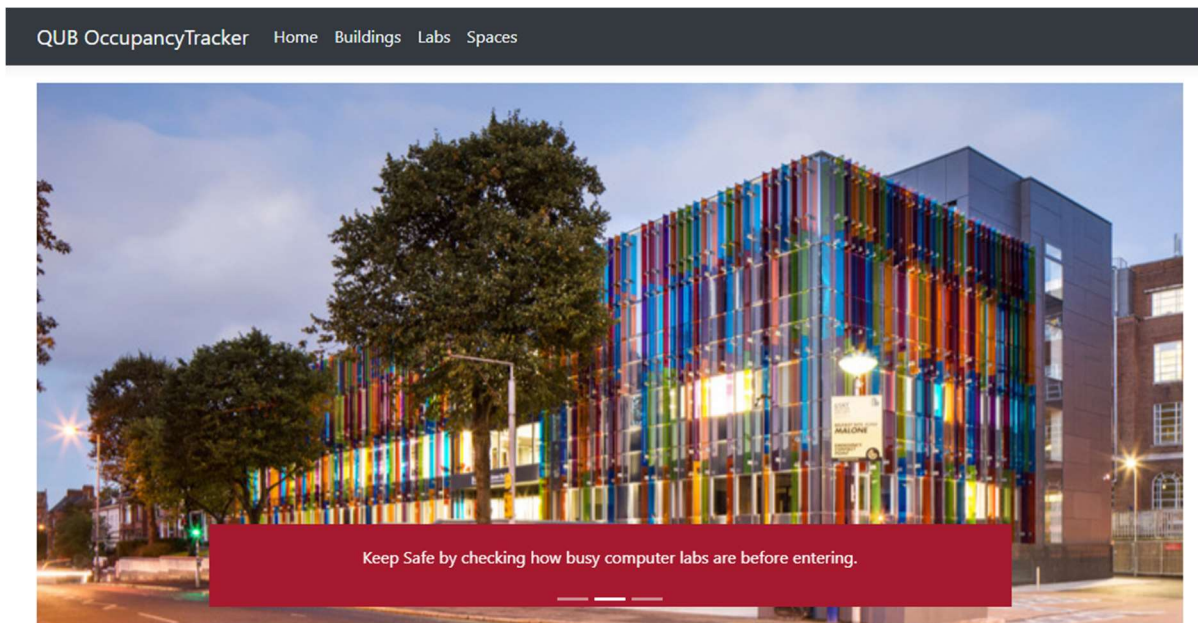


Figure 6: OccupancyTracker Home Page

QUB OccupancyTracker Home Buildings Labs Spaces	
Buildings Occupancy	
Building	Occupancy
Computer Science Building	235
David Keir Building	401
Fake Test Building	0

Figure 7: Buildings Page

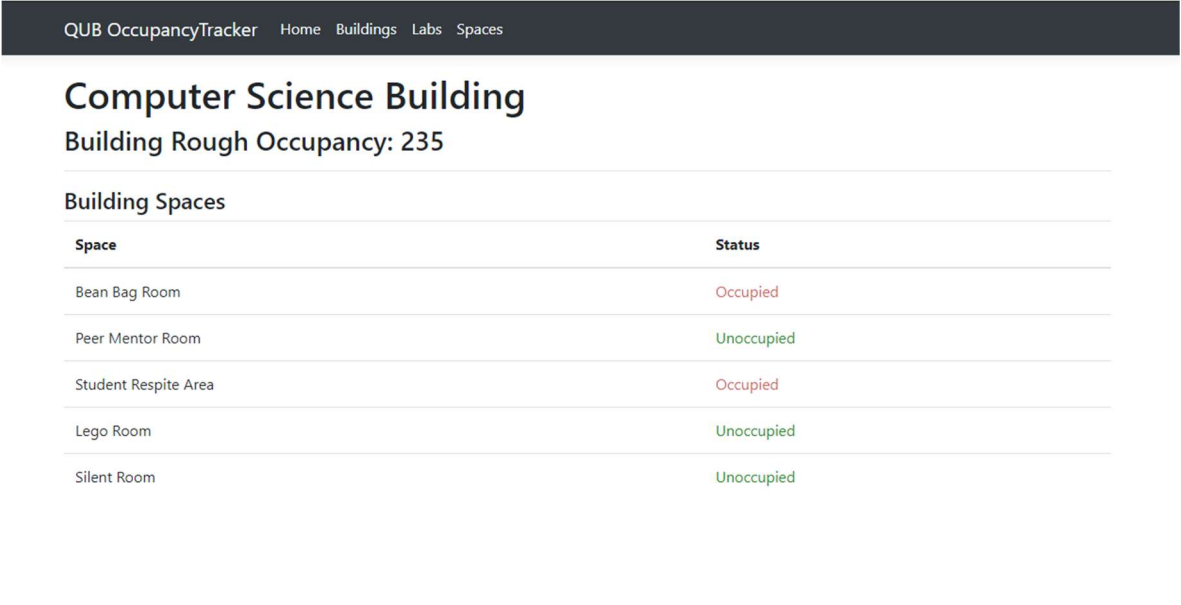


Figure 8: Building Details Page

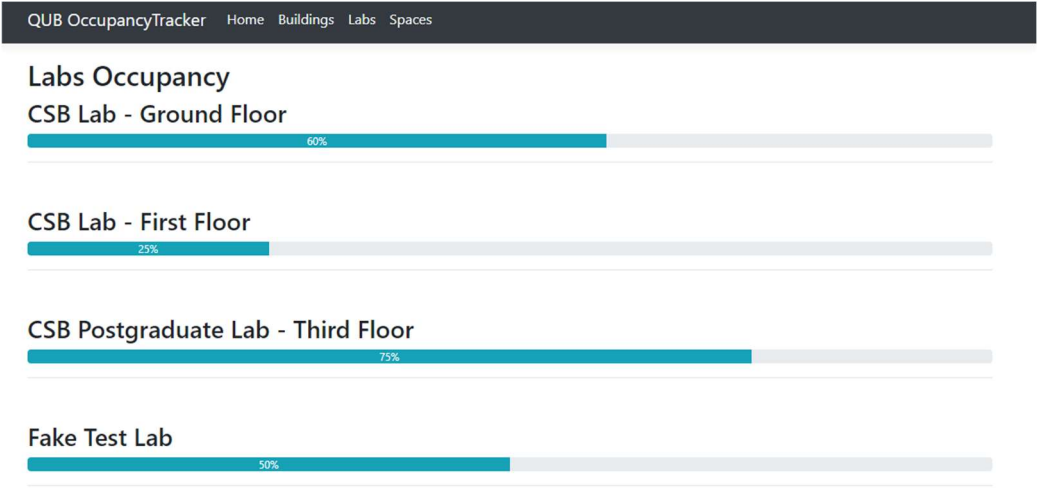


Figure 9: Labs Page

Spaces Occupancy

Room	Status	Building
Bean Bag Room	Occupied	Computer Science Building
Peer Mentor Room	Unoccupied	Computer Science Building
Student Respite Area	Occupied	Computer Science Building
Lego Room	Unoccupied	Computer Science Building
Silent Room	Unoccupied	Computer Science Building
DKB Hub	Occupied	David Keir Building
Chemistry Lab 1	Occupied	David Keir Building
Chemistry Lab 2	Unoccupied	David Keir Building
Psychology Reception	Occupied	David Keir Building
Fake Test Space	Unoccupied	Fake Test Building


Figure 10: Spaces Page

Appendix 4 – Self Report Screenshots

Welcome to the Covid Self-Reporting Portal

Please fill out the form below to submit your details.

Student Number

Date of Covid Diagnosis 

Contact Phone Number

Submit

This information may be used to alert other students you may have been in contact with, but your personal information will never be disclosed.

Figure 11: Self-Report page

Report Submission Successful

Thank you for you self-reporting. Please ensure to self-isolate.

Figure 12: Successful Submission page

Appendix 5 – Entrance System Screenshots

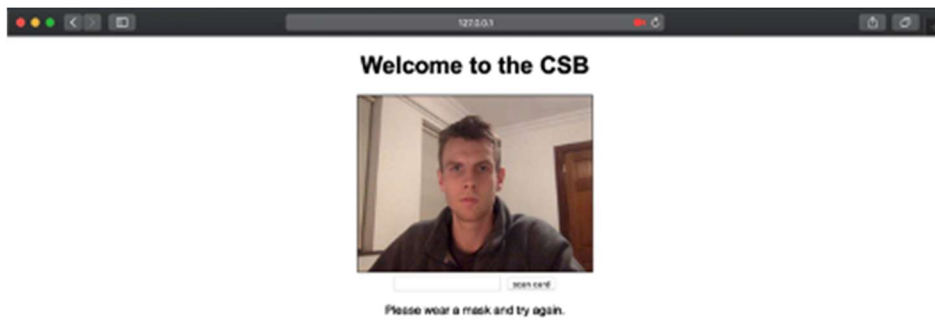


Figure 13 Entrance System denying entry due to not wearing a mask

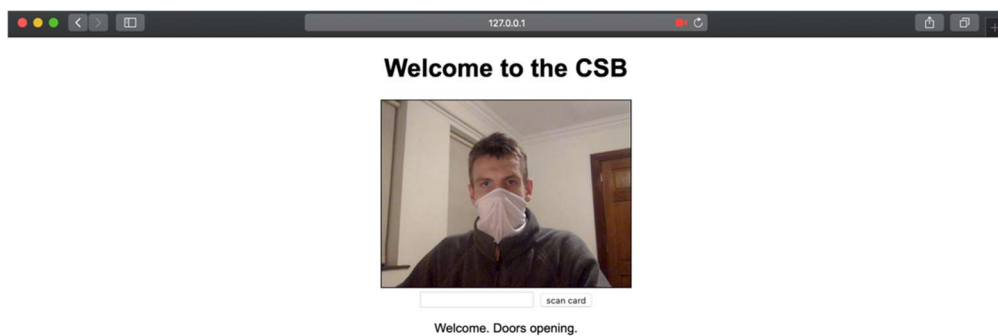


Figure 14 Entrance System accepting entry

Appendix 6 – Facemask Detector

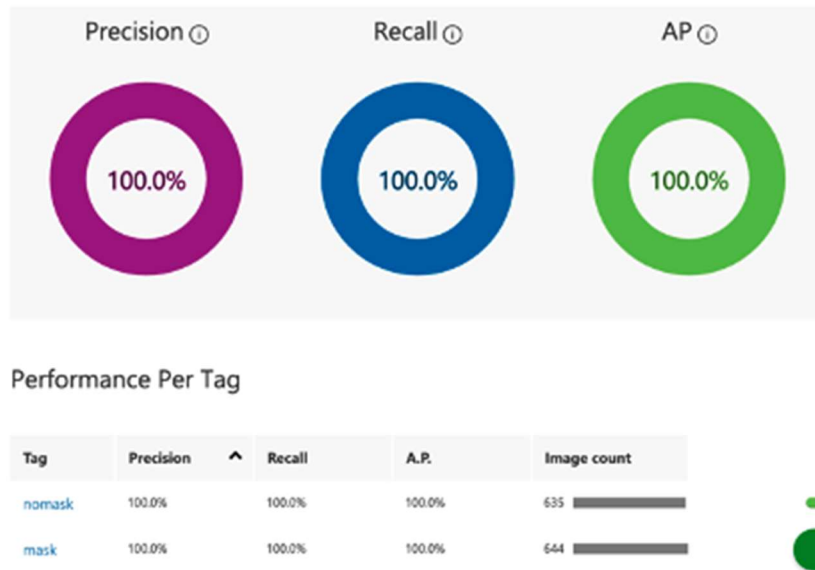


Figure 15 Facemask detector performance statistics

Appendix 7 – API Endpoints

HTTP Method	Endpoint	Description
GET	buildings	Get all buildings
GET	buildings/{id}	Get Building by id
GET	spaces	Get All Spaces
GET	spaces/{id}	Get Space by id
GET	spaces?buildingId={buildingId}	Get all spaces in a building
GET	spaces/occupied	Get all occupied spaces
GET	spaces/unoccupied	Get all unoccupied spaces
GET	spaces/occupied?buildingId={buildingId}	Get all occupied spaces in a building
GET	spaces/unoccupied?buildingId={buildingId}	Get all unoccupied spaces in a building
GET	computers	Get all computers
GET	computers/{id}	Get computer by id
GET	computers/loggedon	Get all logged on computers
GET	computers/loggedoff	Get all logged off computers
GET	labs	Get all labs
GET	labs/{id}	Get lab by id
GET	labs/{id}/computers	Get all computers in a lab
GET	labs/{id}/occupancy	Get lab occupancy
PUT	buildings/{id}/reset	Reset building occupancy to 0
PUT	buildings/{id}/{occupancy}	Set building occupancy to value

PUT	buildings/{id}/entry	Increment building occupancy by one
PUT	buildings/{id}/exit	Decrement building occupancy by one
PUT	spaces/{id}/occupied	Set space status to occupied
PUT	spaces/{id}/unoccupied	Set space status to unoccupied
PUT	computers/logon?id={id}	Set computer to logged on
PUT	computers/logoff?id={id}	Set computer to logged off
PUT	labs/{id}/computers?computerId={computerId}	Add a computer to a lab
PUT	computers/logon?machineName={machineName}	Set computer to logged on via its name
PUT	computers/logoff?machineName={machineName}	Set computers to logged off via its name
POST	buildings	Add new building to system
POST	spaces	Add new space to system
POST	computers	Add new computer to system
POST	labs	Add new lab to system