Mid Term Delivery

Eoin Francis - 21353371
Harsh Saini - 21350903

Table Of Contents

1. Introduction

1.1 Overview

The application we are developing is best described as a lab tutor queueing system, it allows students to request for help from a lab tutor when working in a computer lab. Users will create an account for our system with their username, email, and password. When making a help request students will provide their PC number (so the lab tutors can identify them in the lab) and a description of their request. Once a student has sent their request, they will be notified when it has been accepted by a lab tutor through their student dashboard. Tutors can access a list of all help requests they are assigned to and which have been completed. We have found from our own experience, from our peers, and the Lab Teaching Assistants that all students and tutors face the following issues during lab environments:

- To request help from a lab tutor, students need to raise their hand to get the tutors attention and make them aware of their request. This leads to students keeping their hand raised for long periods of time while the tutor is helping another student, and sometimes the tutor cannot keep track of who needs help next. This results in students not getting the help they need on time when it is their turn. Keeping their hand raised also prevents them from continuing with their work or continuing to focus on the problem themself.
- Another issue faced by students due to long wait times is demotivation. Since the time students spend waiting to be assisted during labs is so long they are demotivated to request for help and in some cases not to attend labs as a whole since they can complete the tasks at home or outside the lab at their own pace.
- An issue lab tutors have is managing the help requests students raise and the order they're raised. Tutors also don't know how long helping a student may take as each student and help request is unique. Often tutors are more focused on managing the help requests when the labs are busy and there is less time to actually help students.

Our project aims to address these issues and make lab environments more practical and efficient for students and tutors.

2.1 Value Proposition

The system streamlines how students request help from lab tutors, eliminating the need to raise their hand. Students can focus on other tasks or attempt to resolve their issues while waiting for assistance.

Students receive instant notifications when a tutor accepts their request or is ready to assist.

Analytics track help request frequency per lab, optimizing tutor distribution and reducing stress caused by high demand. This data also aids in rostering tutors during peak times, saving faculty costs during low-demand periods.
Frequently asked questions will be added to an FAQ page will address common issues, further easing the burden on lab tutors.

2.2 Customer Segments and Target Users

University faculty managers are our primary customer segment. They oversee budgets and resource allocation, including human resources like lab tutors, and ultimately decide on implementing systems like ours. Our solution enhances student performance, improving the faculty's reputation and providing a strong incentive for adoption.

Our target audience includes university students. To understand their needs, we distributed surveys aimed at identifying pain points in the current system. The survey results will guide our design to address their specific challenges. Based on our experience as students, delays in receiving help are common. We also researched a similar system at the University of Illinois-Urbana, which effectively resolved issues around queue management and prioritization.**[1]**

Tutors benefit from the system through an organized list of requests, clearly identified by PC numbers, allowing them to locate students quickly. This eliminates the need to recall which student raised their hand earlier, streamlining their workload and improving efficiency.

2.3 Market Size

The target market for our system comprises all universities with computer labs.
The number of students enrolled in Information and Computer Technology (ICT) courses in Ireland has increased from 15,590 in 2017/2018 to 21,195 in 2023/2024. This provides us with a growing user base who will benefit from the system when it comes to development. **[2]**
The EdTech market is increasing due to a rising need for technologies to help with accessibility and engagement in education. This aligns with our system's aims to improve access for students to help from lab tutors.
We aim to more specifically target university computer labs located within computing schools such as DCU's School of Computing. Courses within these schools will include frequent usage of computer labs for practical classes and lab exams.

2.4 Revenue Streams

The revenue streams for this project are centred on a standard charge which is an annual fee and a subscription fee. The standard charge is €1,500 per annum per lab and provides access to the system's core features. There is also a subscription plan charged at €400 per month per lab. This provides more advanced features in our system. These include 24/7 support and advanced analytics to track lab usage. We expect that we will have two universities in the subscription plan so they can test its features on their busy labs and can implement it in other labs, especially in November and December as lab exams are held and students complete more assignments. This ensures stable, recurring revenue while offering flexibility for institutions with varying needs. A subscription also allows for predictable income and scalable growth as more institutions adopt the system.

3. Scenarios and Constraints

3.1 Operational Scenarios

Home page

1. **Purpose:**
   The home page acts as our system's gateway. It provides navigation options for users to sign up, log in, or learn more about the system.

Links:

- **Register:** Redirects new users to the registration page to create an account.
- **Login:** Directs returning users to the login page to access their account.
- **About:** Provides information about the purpose and functionality of the system.

Scenario 1: User Registration

1. The user visits the home page and clicks on the Register link.
2. The registration page prompts the user to input these details:
    - Username
    - Email
    - Password (and confirmation)
3. Once the form is filled out and submitted:
    - Successful registration: The user is redirected to the respective dashboard based on their user type.
    - Unsuccessful registration (e.g. missing fields or invalid email): An error message is displayed highlighting the issue.

Scenario 2: User Login

1. The user clicks on the **Login** link from the home page.
2. The login page prompts the user to enter their username and password.
3. Once the form is submitted:
    - **Successful Login**: The system validates the credentials. The user is redirected to their respective dashboard:
        - **Student Dashboard**: Displays options for creating and managing help requests.
        - **Tutor Dashboard**: Displays options for viewing and managing assigned help requests.
    - **Unsuccessful Login**:
        - If the username or password is incorrect, an error message appears: "Invalid username or password."
        - The user is prompted to try again or use the **Forgot Password** link if needed.

Scenario 3: Submitting a Help Request (Student User)

- A logged-in student is redirected to their dashboard, which displays the following options:
    - **Create New Request**: Allows the user to fill out a form with:
        - Lab PC Number
        - Description of the issue
    - **View Existing Requests**: Lists pending and completed requests.
- Once a request is submitted:
    - **Successful Submission**: The request is queued, and the student is notified of their position in the queue.
    - **Unsuccessful Submission**: The system highlights errors (e.g., missing fields).

Scenario 4: Managing Requests (Tutor User)

- A logged-in tutor is redirected to their dashboard, which displays the following options:
    - **View Active Requests**: Lists all requests assigned to the tutor.
    - **Accept Request**: Marks the request as being addressed.
    - **Mark as Completed**: Marks the request as resolved.
- When interacting with a request:
    - Tutors can view the student's details, such as their PC number and a description of the issue.

Scenario 5: Error Handling

- **Invalid Links**: If the user attempts to access a restricted page without logging in, they are redirected to the home page with the message: "Please log in to continue."
- **Page Not Found**: If a user accesses a non-existent URL, they are shown a 404 error page with options to return to the home page.

Purpose of Each Link:

1. **Register**: Directs users to create an account, enabling them to use the system.
2. **Login**: Redirects to the login page to validate users' credentials and grant access.
3. **Create New Request**: Enables students to submit help requests.
4. **View Existing Requests**: Allows users to track the status of their requests.
5. **Accept Request (Tutor)**: Assigns a request to the tutor, indicating it's being resolved.
6. **Mark as Completed (Tutor)**: Updates the request status to be resolved.

Constraints/Challenges

Time

We aim to have a working prototype finished by the middle of March to demonstrate within an actual lab session. We will need to ensure that the core functionality of the system works by this time and that the most requested features from our initial survey are implemented. We will also need to manage this project alongside our other assignments. We have created a Trello board for this project to ensure that we remain aware of our tasks and deadlines.

Choice of Features

We need to consider which features are most realistic for our system. For example, some students may like us to implement a feature that does not return a lot of results in our initial survey. They may feel like their opinion does not matter, but it will not be possible for us to fully satisfy every respondent.

We will also need to consider the impact each feature may have on tutors and their performance. They need to feel the usefulness of our system in order for them to perform effectively.

4. Summary Functional Specification

4.1 General Description

Our system is a **help queue management system** tailored for educational institutions. It will allow students to submit requests for assistance during lab sessions and tutors to efficiently manage and respond to them. The system's architecture will ensure a seamless user experience, with real-time updates and automated queue management. The platform will be built using Django for backend logic and Channels for WebSocket-based real-time communication.

We have developed several diagrams to represent our architecture:

- **Basic System Architecture Diagram:** Illustrates the core components, such as client interfaces, application logic, and the database layer, and their interactions.
- **High-Level Architecture Diagram:** Depicts the deployment of our system, including hosting on AWS and the integration of ASGI (Daphne server) for real-time requests.
- **Entity Relationship Diagram (ERD):** Displays relationships between core data entities like Student, Tutor, and HelpRequest.

Challenges

Some challenges we foresee include:

1. Ensuring real-time updates without performance degradation, especially as the system scales.
2. Implementing an efficient queueing system that evenly distributes help requests among tutors while accounting for tutor availability.
3. Managing user authentication and roles, particularly ensuring secure role-based access for students and tutors.
4. Designing a user-friendly interface that caters to both students and tutors with minimal learning curve.

Initial Functional Requirements

3.1 Registration

Description:
This is the first step a user completes to use our system. Upon loading the home page, users will find a "Register" option in the navbar. Here, they will provide their username, email address, and password.

Criticality:
This function is necessary to distinguish between students and tutors and ensure authenticated access to the system's features.

Technical Issues:
The registration page will be created using **HTML**, **CSS**, and **Bootstrap** for styling. Django's authentication system will manage the user database and ensure secure password hashing.

Dependencies:

- Django's built-in User model for authentication.
- Middleware for session and CSRF protection.

3.2 Log In

Description:
Allows users to log into the system by providing their registered username and password. The system will validate their credentials against the database and log them in if the details match.

Criticality:
Logging in is essential for students and tutors to access their respective dashboards and interact with the system.

Technical Issues:

- Implementing secure login with Django's authentication framework.
- Handling login sessions and role-based redirection (student dashboard for students and tutor dashboard for tutors).

Dependencies:

- Django's session middleware for managing user sessions.
- CSRF tokens for form security.

3.3 Create And Send Request

Description:
Allows students to create a help request for lab assistance. The form will require the student's PC number and a description of the issue. Once submitted, the request will be queued and visible to tutors in real-time.

Criticality:
This is the core function of the system, enabling students to request help and tutors to manage their workload.

Technical Issues:

- Ensuring real-time updates of new requests on the tutor dashboard using Django Channels.
- Validating form inputs to ensure accurate information is submitted.

Dependencies:

- Django Channels for WebSocket communication.
- HelpRequest model for database interaction.

3.4 Modify Requests

Description:
Allows students to manage their submitted help requests. They can cancel their requests if no longer needed, or mark them as resolved if help was received outside the system. Tutors can update request statuses to reflect progress, such as marking requests as "In Progress" or "Completed."

Criticality:
Ensures that the system remains dynamic and reflects real-time updates in the queue.

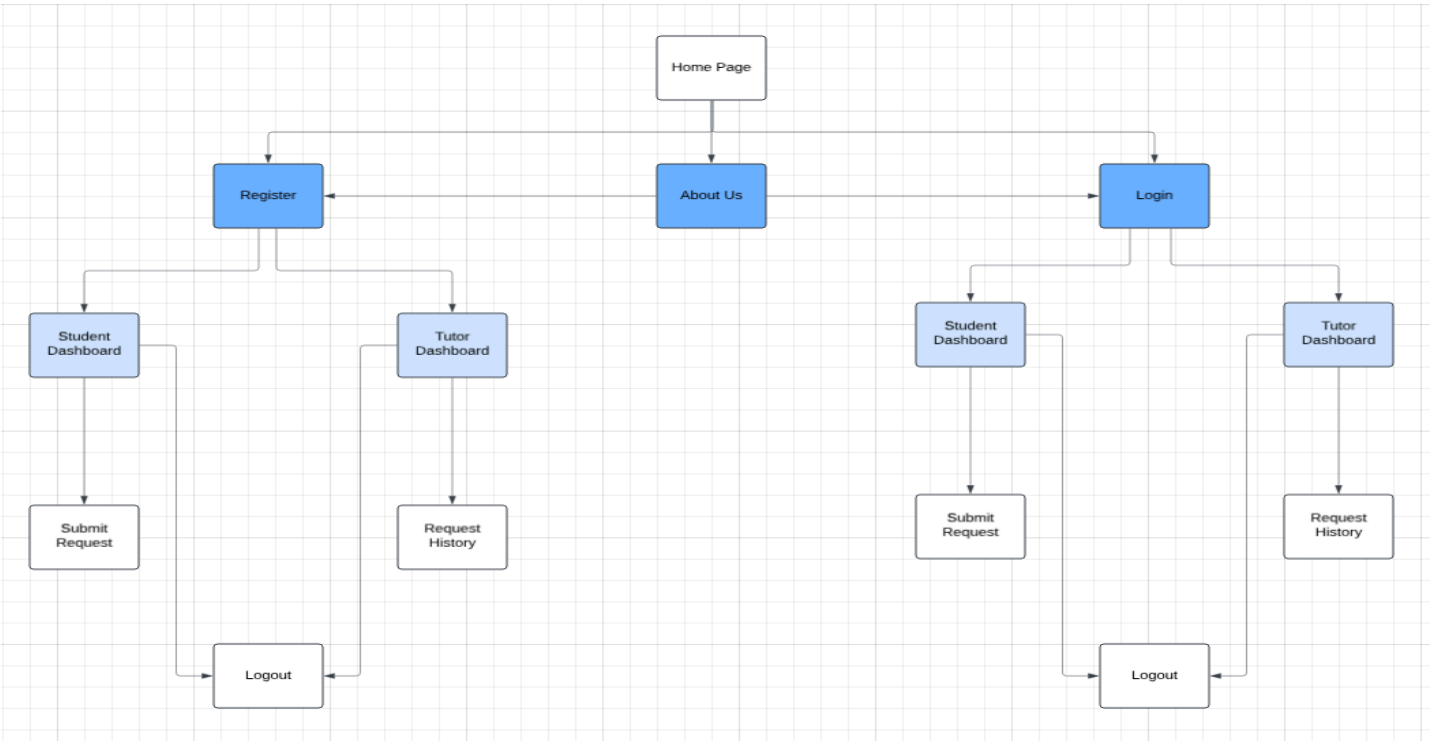
Technical Issues:

- Managing real-time updates when requests are modified.
- Ensuring that only authorized users (students or tutors) can modify a request.

Dependencies:

- WebSocket Consumers to broadcast updates.
- Role-based access control for request modification.

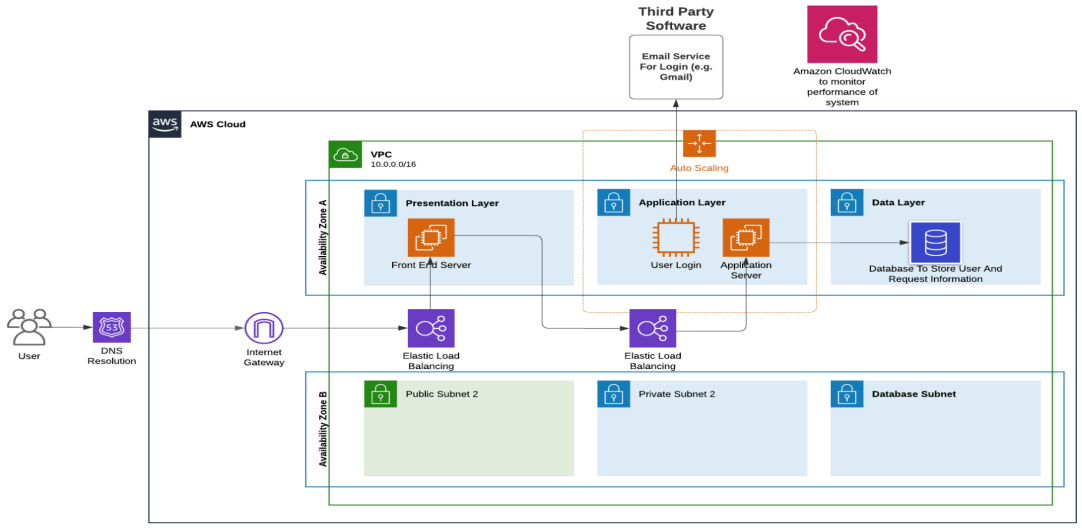
4.2 Sitemap Diagram



Sitemap Diagram

4.3 AWS Diagram

[AWS Diagram: Lucidchart]



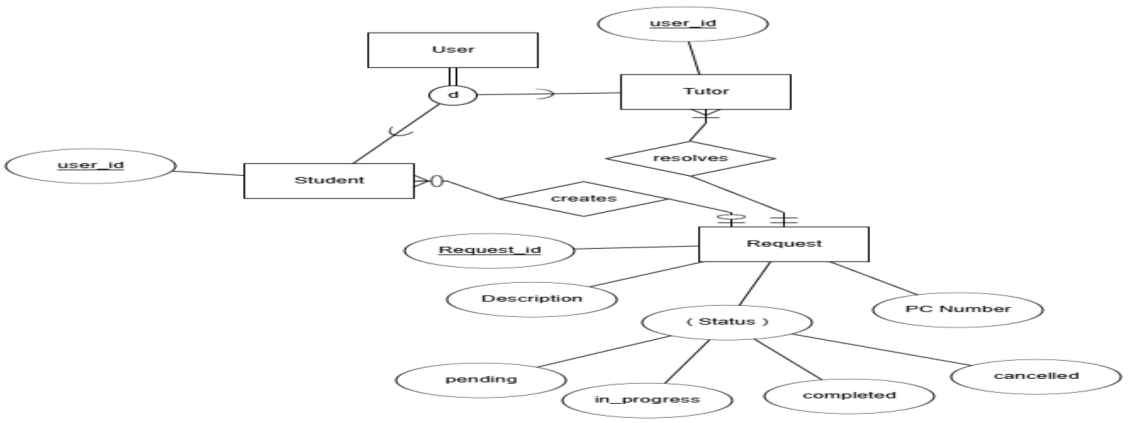
- **Internet Gateway:** Allows for communication between our VPC and the internet.
- **Domain Name Server (DNS):** A DNS is used by the web browser to help identify the web server's IP address before the user's request is sent.
- **Load Balancer:** Provides scalability for the system at times of high usage by evenly distributing web traffic across EC2 instances.
- **Auto Scaling Groups (ASG):** To scale the Django servers based on load.
- **EC2 Instances:** To host our Django app
- **Amazon RDS:** For storage of data including data relating to our Django models.
- **Route 53:** To manage the domain name for our system.
- **Cloudwatch:** To analyse the performance of AWS services.

Microservices:

- **User Login:** Will utilise an API to process user login, such as Gmail.

[3] [4]

4.4 ER Diagram



Description

Our ER diagram contains two entities, a request entity and a supertype user entity, which contains a student and tutor entity. Both of these subtype entities contain a unique 'user_id' attribute.

The request entity includes a unique ‘request_id’ attribute, ‘description’, a composite ‘status’ attribute and ‘PC Number’. The composite ‘status’ attribute includes it’s own four attributes, ‘pending’, ‘in_progress’, ‘completed’ and ‘cancelled’.
The relationships between entities include students who create requests and tutors who resolve requests.

4.5 Challenges To Be Overcome During Software Development

Some challenges we foresee include:

1. **Real-Time Updates and Performance:**
    - Implementing real-time updates via WebSockets or Django Channels is technically demanding. Ensuring the system remains responsive under increasing user loads will be a key challenge, especially when scaling to handle multiple concurrent requests simultaneously.
2. **Efficient Queue Management:**
    - Designing and implementing a queueing mechanism that assigns requests to tutors fairly and efficiently while accounting for tutor availability is complex. Handling cases like a tutor becoming unavailable mid-session or students canceling requests requires robust logic.
3. **Secure User Authentication and Role Management:**
    - Developing a secure authentication system with role-based access control to distinguish between student and tutor functionalities is critical. Protecting sensitive data and preventing unauthorized access to role-specific features adds more complexity.
4. **User-Friendly Interface:**
    - Creating an accessible interface for all users, with a minimal learning curve, requires careful design and feedback from potential users. Ensuring usability while maintaining functionality is essential to the project's success.

4.6 What Parts Of Idea Will Be Prepared For Prototype

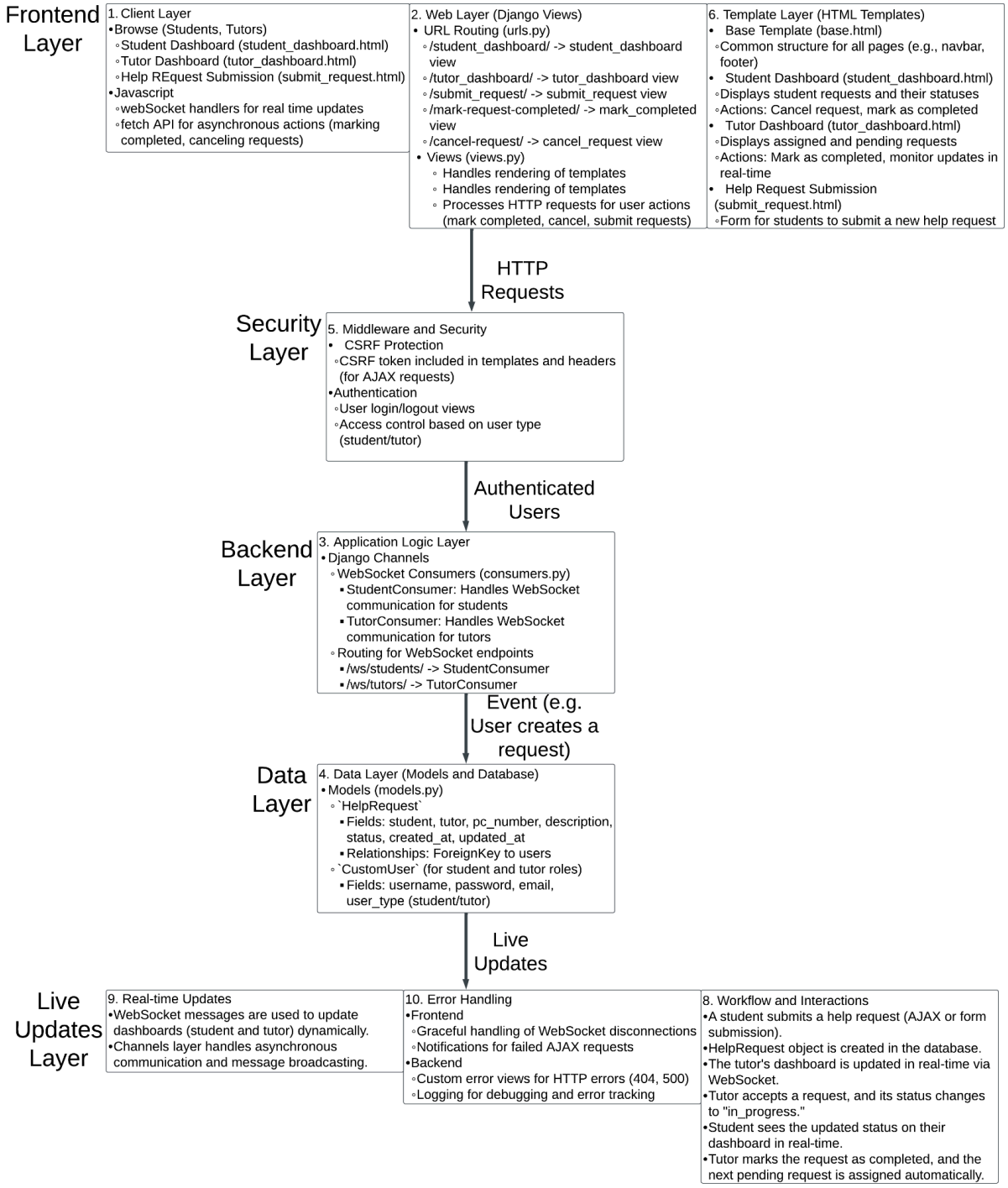
For the prototype, we plan to deliver:

1. **Student and Tutor Dashboards:**
    - Fully functional dashboards tailored for each user type, allowing students to submit and track requests and tutors to manage and respond to assigned requests.
2. **Real-Time Communication:**
    - Implementation of WebSocket-based functionality for live request tracking, real-time notifications, and updates on both dashboards.
3. **User Authentication System:**
    - Basic authentication allowing users to register, log in, and access role-specific features. This will include role management to ensure secure access for students and tutors.
4. **Help Request Management:**

- Functionality for students to create, cancel, and view their requests. Tutors will have the ability to accept requests, mark them as completed, and view pending requests in their queue.

5. **AWS Deployment:**
   - Hosting the prototype on AWS to enable live demonstration and testing. This includes deploying the backend, database, and frontend components, ensuring seamless functionality in a live environment.

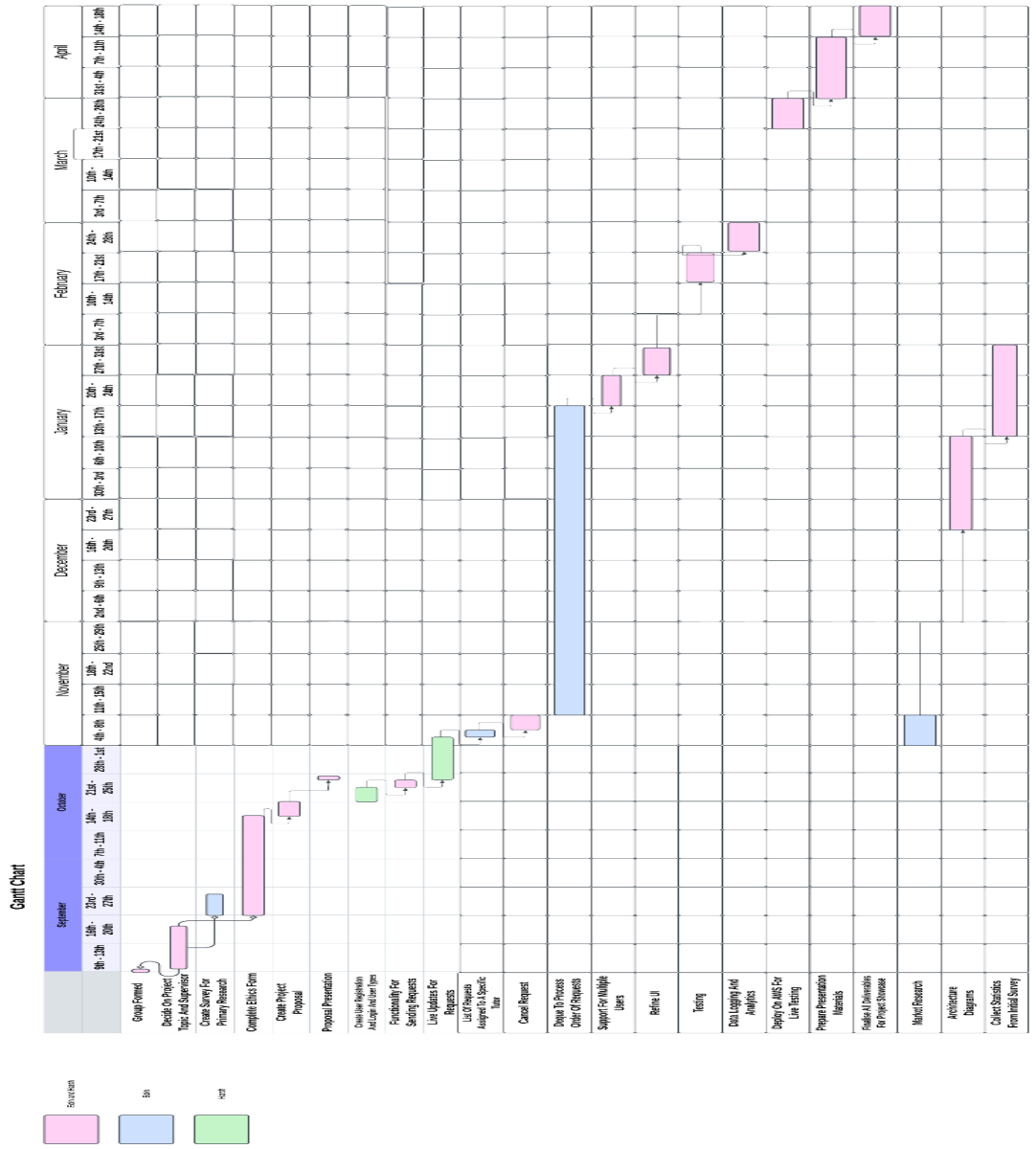
4.7 System Architecture



The front end/client side will include the website. This is where the user will interact with our system and connect to the Django app through WebSocket, allowing for live updates. It is created using HTML, CSS and Bootstrap.
An Asynchronous Gateway Server Interface or ASGI will handle asynchronous events through WebSocket. This contrasts with the WSGI that comes with a standard Django application that can only handle synchronous events. This functionality for example allows a student to be notified when a tutor has chosen their request without the need to frequently refresh the page. Daphne is used in our system for ASGI.**[5]**
In addition, Django Channels can handle WebSocket requests. Channels are similar to pipelines where a message can be sent from one location to another.
Each channel can contain consumers which are Channel's equivalent to Django's views. They can contain functions which can be called when needed.
Our application uses one consumer for the dashboard, comprising both students and tutors. This is defined within a routing.py file.
The application logic layer will manage the real-time functionality of the system through Django Channels. This is provided through WebSocket consumers which dynamically updates both the student and tutor dashboards using JavaScript.**[6]**
An SQLite database stores information in our system. This includes information relating to the Django models defined for users and requests.

5.1 Gantt Chart

Link to Gantt Chart:
https://lucid.app/lucidchart/de69abf8-b992-4f4a-8d54-b33660530dc9/edit?invitationId=inv_d012763c-8132-4fc3-8e55-959de5b2eedf



Also see appendix for a larger view of the Gantt chart

5.2 Timeline

Initial Stages: Idea Formation and Ethics Approval

- **9th September**: **Group Formation**
  Our team was formed based on our successful collaboration in previous projects. We identified our strengths and skills, which we believed would enable us to execute a technically challenging yet innovative project effectively.
- **19th September**: **Project Topic and Supervisor Selection**
  After brainstorming ideas, we decided on a **lab tutor queueing system** to address inefficiencies in the existing processes. We chose this idea because it aligned with our interests and experience, and has the potential for a significant real-world impact.
- **16th October**: **Ethics Form Completion**
  The ethics form was submitted, ensuring our project complied with GDPR and institutional ethics policies, particularly as we planned to gather feedback from research.

Planning and Proposal Development

- **18th October**: **Proposal Document Completion**
  The project proposal was finalized, outlining the scope, objectives, and deliverables for our system. It included a breakdown of tasks and a rough timeline for development.
- **25th October**: **Proposal Presentation**
  We presented our project idea to a panel, receiving valuable feedback that we used to refine our approach and identify potential challenges.

Development Phases

- **November - December**: **Initial Development Milestones**
  - **User Registration and Login** (Harsh): Implemented a secure system to allow students and tutors to register and access role-specific dashboards.
  - **Single Request Functionality** (Harsh/Eoin): Developed the basic feature allowing one student to send a help request to one tutor, ensuring the foundational functionality of the system was operational.
  - **List of Requests** (Eoin): Created an interface enabling tutors to view pending, resolved, and completed requests, improving user experience.
  - **Live Updates for Requests** (Harsh): Introduced WebSocket functionality to provide real-time notifications for students and tutors. This eliminated the need for manual page refreshes.
  - **Cancel Request Feature** (Harsh): Added functionality for students to cancel help requests if resolved independently.

Research and Design

- **October - November**:
  - **User Surveys** (Eoin): Designed and distributed surveys to second-year students to understand their pain points and assess the effectiveness of proposed solutions.
  - **Market Research** (Eoin): Investigated the potential market for the system, focusing on educational institutions and the scalability of the solution.
- **December**
  - **Architecture Diagram Design** (Both): Created detailed diagrams illustrating the high-level system architecture, data flow, and interactions between system components.

Upcoming Milestones

January 2025:

- Expand system functionality to support **multiple concurrent users** (students and tutors).
- Implement **load balancing** for help request distribution among tutors.
- Refine the user interface for an intuitive and efficient user experience.

February 2025:

- Conduct **extensive testing**, including functional, load, and usability tests.
- Implement **data logging and analytics** for system performance monitoring.

March 2025:

- Deploy the prototype on **AWS** for live testing and feedback collection.
- Prepare **presentation materials** for the final expo, including a working prototype, diagrams, and a detailed report.

April 2025:

- Finalize all deliverables for submission and the **project showcase**.
- Conduct final rehearsals for the expo presentation.

6. Appendix

Appendix A: Surveys

1. **Purpose**: Surveys were distributed to second-year students to identify challenges faced in lab environments and gather insights for system design.
2. **Key Findings**:
    - Students frequently face long wait times, causing frustration and demotivation.
    - Tutors struggle with managing multiple help requests efficiently.
3. **Survey Structure**:
    - **Survey 1**: Focused on understanding student challenges.
    - **Survey 2**: Evaluated how well the proposed system meets student needs.

Appendix B: Diagrams

1. **Basic System Architecture Diagram**:
    - Describes interactions between the client (students and tutors), application server (Django with Channels), and database (PostgreSQL).
2. **High-Level Architecture Diagram**:
    - Shows deployment on AWS, highlighting EC2 instances, RDS, S3, and Route 53 integration.
3. **Entity Relationship Diagram (ERD)**:
    - Displays relationships between students, tutors, and help requests, emphasizing core attributes and connections.

Appendix C: Gantt Chart

- A detailed Gantt chart was developed to outline task schedules and responsibilities, including development milestones, research phases, and testing timelines.
- Link to Gantt Chart:

Appendix D: References

1. **Academic References**:
    - Adoption of an Online Queue App for Higher Education: A Case Study – ASEE PEER.

    - HEA Key Facts and Figures – Enrolment Trends.
2. **Technical References**:
    - AWS Architecture and Services Overview.
    - ASGI and Django Channels Implementation Guides.

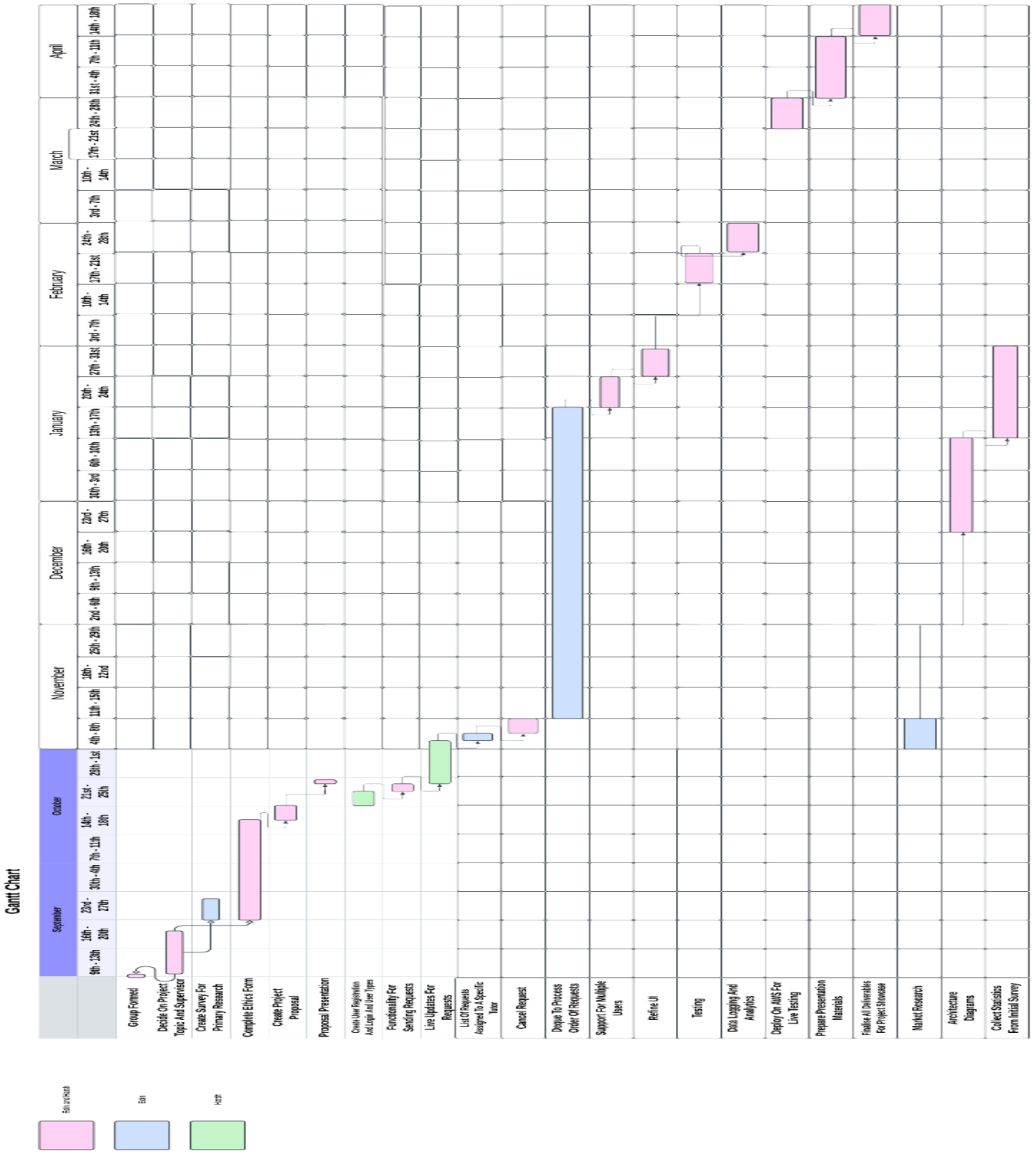
Appendix E: Additional Resources

1. **Trello Board**: A Trello board was used for task management and deadline tracking.
2. **Lucidchart Diagrams**:
    - System architecture and AWS deployment diagrams were created using Lucidchart for clarity and accuracy.

Appendix F: Technical Dependencies

1. **Software**:
    - Django Framework for backend development.
    - PostgreSQL for data storage.
    - Bootstrap for UI design.
2. **Hosting**:
    - AWS EC2 and RDS for deployment.
3. **Real-Time Communication**:
    - Django Channels and Daphne server for WebSocket support.

Appendix G: Gantt Chart



Gantt Chart Key:

Pink = Eoin and Harsh
Blue = Eoin
Green = Harsh

Appendix H: Student Declaration of Academic Integrity

Students may be required to submit work for assessment in a variety of means, for example physical submission or electronic submission as per the lecturer's instructions.

In all cases students must make a declaration of academic integrity, either by physically completing such a declaration and submitting it with their assignment or engaging appropriately with the electronic version of the declaration. Assignments submitted such that the form has not been included, or the electronic equivalent has been circumvented, will not be accepted.

Declaration

Name:	Eoin Francis, Harsh Saini
Student ID number	Eoin Francis - 21353371, Harsh Saini - 21350903
Programme	COMBUS4 – Computing for Business 4
Module Code	CSC1118
Assignment Title	Mid Term Delivery
Submission Date	17/01/2025

I understand that the University regards breaches of academic integrity and plagiarism as grave and serious.

I have read and understood the DCU Academic Integrity and Plagiarism Policy. I accept the penalties that may be imposed should I engage in practice or practices that breach this policy.

I have identified and included the source of all facts, ideas, opinions and viewpoints of others in the assignment references. Direct quotations, paraphrasing, discussion of ideas from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the sources cited are identified in the assignment references.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

I have used the DCU library referencing guidelines (available at https://www4.dcu.ie/library/classes_and_tutorials/citingreferencing.shtml **and/or** the appropriate referencing system recommended in the assignment guidelines and/or programme documentation.

By signing this form or by submitting material online I confirm that this assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

By signing this form or by submitting material for assessment online I confirm that I have read and understood the DCU Academic Integrity and Plagiarism Policy (available at http://www.dcu.ie/registry/examinations/index.shtml).

Signature **Eoin Francis, Harsh Saini**_____

Date: 17/01/2025

7. References

[1] Jensen, K., R.Amos, J., Angrave, L., Flanagan, K., Mussalman, D., D. Schmitz, C., Fagen-Ulmschneider, W. (2019, June 15) ***"Adoption of an Online Queue App for Higher Education: A Case Study".*** **ASEE PEER.** ASEE PEER - Adoption of an Online Queue App for Higher Education: A Case Study

[2] (Updated 2024, October 21). ***"Key Facts and Figures"*****. HEA. (Enrolment Trends B)** **https://hea.ie/statistics/data-for-download-and-visualisations/key-facts-figures/**

[3] Taylor, M. (2024, December 19). "*AWS Architecture: Components and Diagram Explained".* Theknowledgeacademy. **https://www.theknowledgeacademy.com/blog/aws-architecture/#:~:text=AWS%20Architecture%20Diagrams%20typically%20include,S3%20buckets%2C%20and%20RDS%20databases.**

[4] Safe. (2023, November 10). ***"Basics of AWS Cloud Computing".*** **AWS.plainenglish.** https://aws.plainenglish.io/basics-of-aws-cloud-architecture-cf949129b824

[5] Yegulalp, S. (2022, July 13). ***"ASGI Explained: The Future Of Python Web Development".*** **InfoWorld.** ASGI explained: The future of Python web development | InfoWorld

[6] ***"Guide to Django Channels: What it is, pros and cons and use cases".*** **Ably.** https://ably.com/topic/what-is-django-channels#:~:text=Django%20Channels%20splits%20any%20incoming,IP%20address%20(for%20WebSocket)