KUZEY KIBRIS KAMPÜSÜ

ODTÜ METU

NORTHERN CYPRUS CAMPUS

CNG 495

CLOUD COMPUTING

FALL 2025

CAPSTONE PROJECT FINAL REPORT

Yıldız Alara Köymen - 2453389

Burak Mirac Dumlu - 2584944

Emir Canlı - 2637544

Table of Contents

1. Introduction

1.1 Project Overview

DynaZOR is a cloud-based scheduling and rendezvous system designed to automate appointment management between users. The goal is to eliminate manual scheduling errors and reduce communication delays by using AWS services for storing data and notifications. The system dynamically adjusts appointments when changes occur and instantly notifies users through cloud based alerts, ensuring efficient coordination and improved time management. The system also has a notification system. When an appointment is cancelled, it provides immediate notifications to all necessary parties. This guarantees that everyone is always up to date, reducing miscommunication and missing meetings. The system will use the collected appointment data to perform simple analytics and show respectively more booked hours and which user books the most. For example, analyzing whether a user's schedule is usually full in the morning or later in the day. The project will be implemented as a Platform as a Service (PaaS) solution. Users will be able to access the system through a web interface without needing to install or manage any local software.

1.2 Benefits and Novelties

DynaZOR improves efficiency and reliability by combining smart scheduling and transparent communication. It eliminates administrative labor, improves time management, and guarantees that all parties are prepared and informed for their appointments. One standout aspect is its clever rescheduling. If an appointment is cancelled or a dispute emerges, the system immediately intervenes. It discovers a new open time that is convenient for everyone depending on their preferences and schedules. This eliminates the need for users to manually calculate new times.

The insights will be displayed to users as a small dashboard or chart, allowing them to see patterns like “You are most likely to be full on at 10 AM.” This way, users can adjust their future habits and create appointment times that fit their routines more reliably.

The system is also self-contained.This is required for secure&offline environments such as military hospitals, internal banking networks, or government facilities where data privacy laws prevent connecting to third-party cloud calendars.

1.3 Related Work & Competitive Analysis

1.3.1 cal.com (https://github.com/calcom/cal.com)

Cal.com is an enterprise-grade, open-source scheduling platform designed to act as an "infrastructure" layer that syncs with external providers like Google Calendar and Outlook to manage global availability, payments, and time zones.

The primary difference is that while Cal.com focuses on broad interoperability and aggregating availability from existing external calendars, our project is a standalone, self-contained backend that generates and enforces its own schedule and inventory logic within a custom SQL database. Additionally, our system implements a specialized priority-based waitlist algorithm with automatic user promotion and denormalized analytics (using counters), a specific resource-allocation feature for high-demand slot management.

1.3.2 https://rallly.co/ (https://github.com/lukevella/rallly)

Rallly is a free, open source scheduling and collaboration tool designed to simplify the process of finding common meeting times through availability polling. Unlike traditional calendar tools that require extensive back and forth communication, Rallly allows users to quickly create scheduling polls and share invite links with participants, who can then vote on proposed dates and times to find the most suitable option for the group.

Even though its scheduling purpose is different from ours, it still serves as a good example of how scheduling algorithms and applications work. There are many aspects such as state management, scheduling, rescheduling, and conditional rendering that need to be considered when working on such an application. Rallly’s repository includes examples using different frontend frameworks, which can be referenced when developing a scheduling application similar to ours.

2. Structure of the Project

2.1 System Overview: Modules and Functions

The system is divided into 3 modules: the Client-Side Application, the Backend API, and the Cloud data service. Each module handles specific responsibilities.

1. The User Interface (Frontend) Module

The User Interface module of the system is implemented as a React application built with Vite and styled primarily using Tailwind utility classes. It communicates with the Flask backend through HTTP requests handled by Axios via a shared API wrapper. Consistency for endpoint routes is handled by the addition of endpoint constants for different api calls. Client side routing is managed using React Router, enabling easy navigation between core screens such as authentication, dashboard, scheduling, analytics, profile management, and administrative utilities. The frontend provides users with an interactive scheduling experience, including availability selection, booking states, queue indicators, and appointment limits, while offering administrators password protected database management tools. State management is handled using React hooks, with loading indicators and feedback messages. Protecting state changes enables the dynamic rendering of the schedule for owners and viewers. The application is designed for static deployment and is configured to consume backend services through environment based API endpoints, making it suitable for cloud hosting platforms such as Vercel.

2. The Application Logic (Backend) Module

The Application Logic (Backend) module is implemented using a Flask based RESTful API and serves as the core processing layer of the system. Acts as the intermediary between the client and the database, enforcing business rules. Manages JWT tokens and session security. Sanitizes incoming JSON data and triggers calculation algorithms. Triggers email notifications via Amazon AWS. The backend is responsible for authentication, scheduling, appointment management, analytics generation, and administrative operations. It interacts with a relational database through a dedicated data access layer and manages core domain entities such as users, schedules, timeslots, waitlists, and appointment statistics. Business logic enforces booking constraints, maintains queue ordering, updates aggregated analytics, and reassigns freed time slots to waiting users when availability changes. In addition, the backend exposes role restricted administrative endpoints for database initialization, modification, and backup. The service is configured through environment variables

and packaged for containerized deployment, allowing it to operate reliably in a cloud hosted environment.

3. The Database Module

Manages persistent data storage and file hosting. Stores relational data (users, orders) in Amazon RDS through Microsoft SQL.

2.2 System Architecture & Information Flow Diagrams

The DynaZOR's system architecture is designed by diagrams and theoretical charts at the start of the project. Then built, changed & integrated the top of the first diagrams that has been created. Therefore, understanding the system via charts is important to understand the architecture.

The system architecture of DynaZOR is designed to be modular, scalable, and cloud-native. Starting from the base structure, we utilize distinct functional modules to communicate via Restful APIs. The infrastructure leverages Amazon RDS for storage, minimizing the need for physical server maintenance & Amazon SNS for notification which reduces cost. Figure 1 below provides a high-level view of how these cloud services interact with the application logic and the end-users.

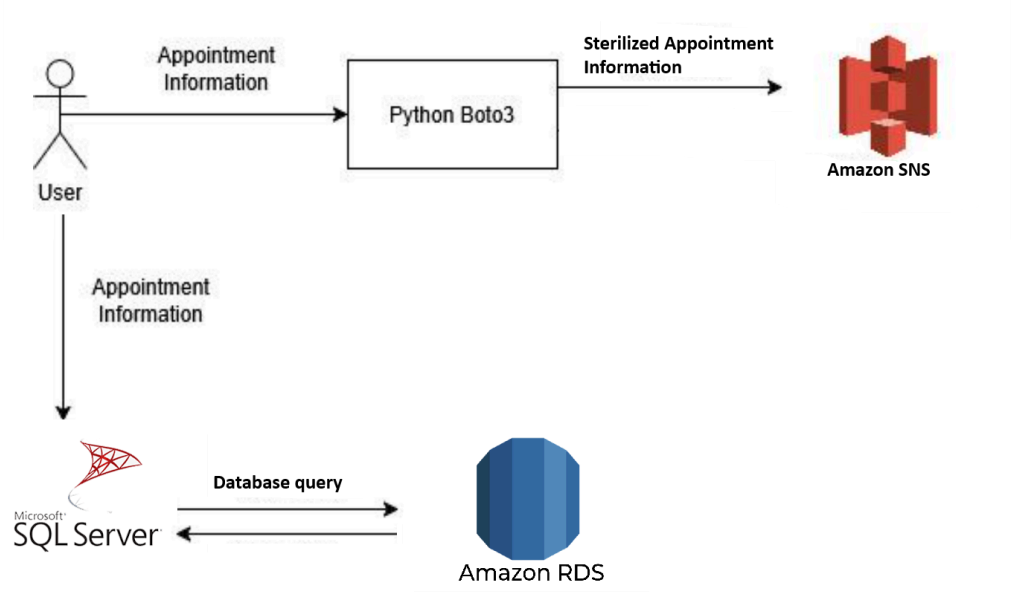

Figure 1: System architecture of DynaZOR

While mentioning database query and Amazon RDS, Figure 2 illustrates the Entity Relationship Diagram (ERD) for the scheduling system.The core entities and their relationships are defined as follows:

- **Users:** The central entity of the system. It stores authentication details and personal information. It acts as the parent entity for almost all other tables,all schedules and appointments are linked to a valid system user.
- **UserSchedule & Timeslots:** These tables handle the core availability logic. A one-to-many relationship exists between users and their schedule, allowing a user to define availability for specific dates. The timeslot table further granulates this by breaking down a schedule into specific hourly/minute intervals, which can be flagged.
- **PriorityQueue:** This entity manages high-demand slots. It utilizes a composite primary key to prevent duplicate queuing for the same slot by the same user. It links users to specific timeslots they are waiting for, ordered by **first-come-first-server** priority order.
- **AppointmentStats:** Designed for analytics, this table tracks the interaction between an "owner" (the person offering the slot) and a "booker" (the client). It uses a composite key including time to record the frequency of bookings between specific user pairs.

Key Relationships:

- **1:N (One-to-Many):** A User can have multiple Schedules; a Schedule has multiple Timeslots.
- **M:N (Many-to-Many):** The queue effectively resolves a many-to-many relationship between Users and Timeslots, allowing multiple users to queue for multiple slots.

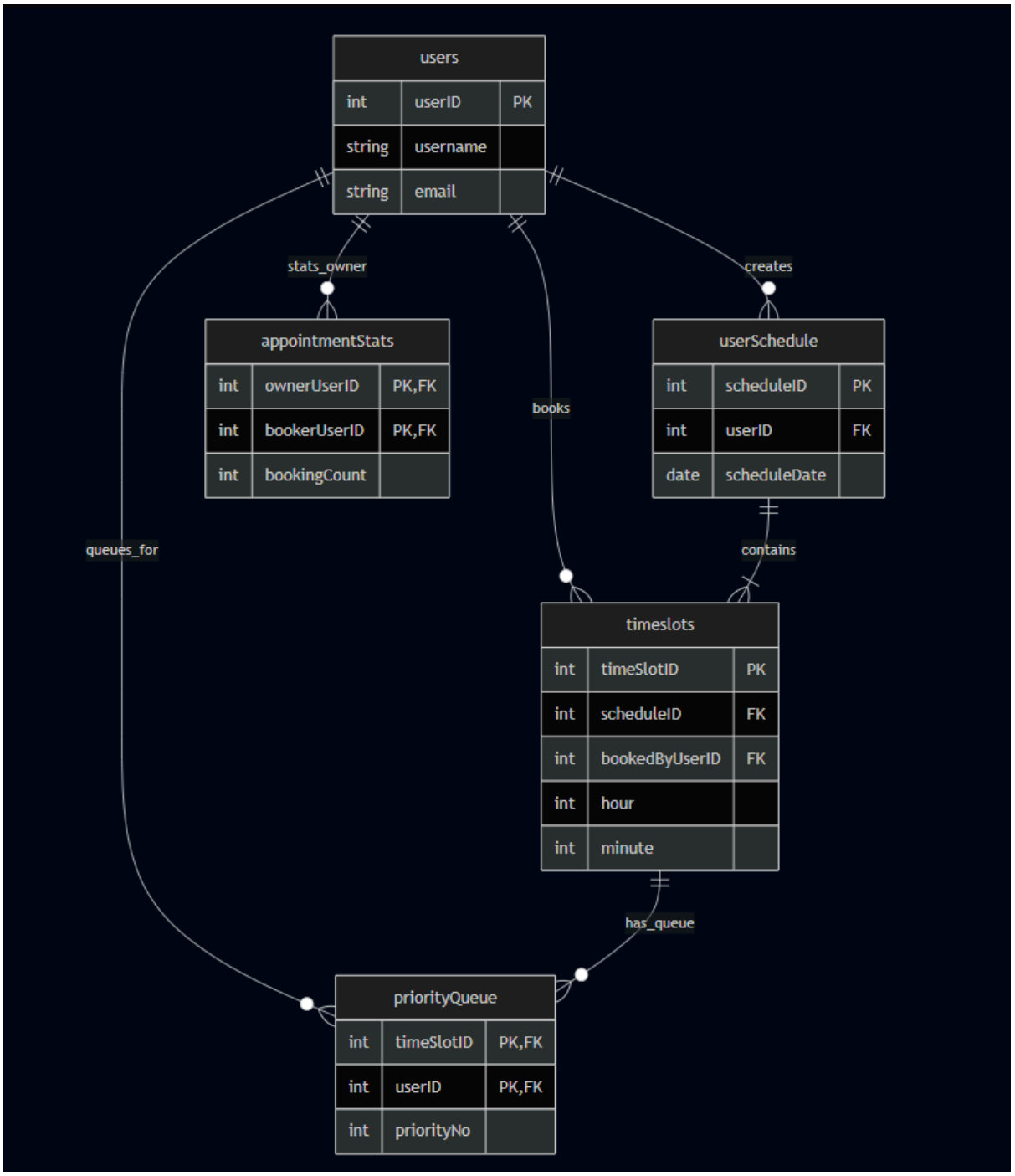

Figure 2: Relational Database Overview

Understanding the base database architecture is crucial for seeing the implications on the overall system which is told below Figure 3. The use case diagram illustrates the main interactions between users and the DynaZOR

system. Its purpose is to show the system's core functionalities, such as creating, updating, rescheduling, and cancelling appointments, and how different user roles interact with these features.

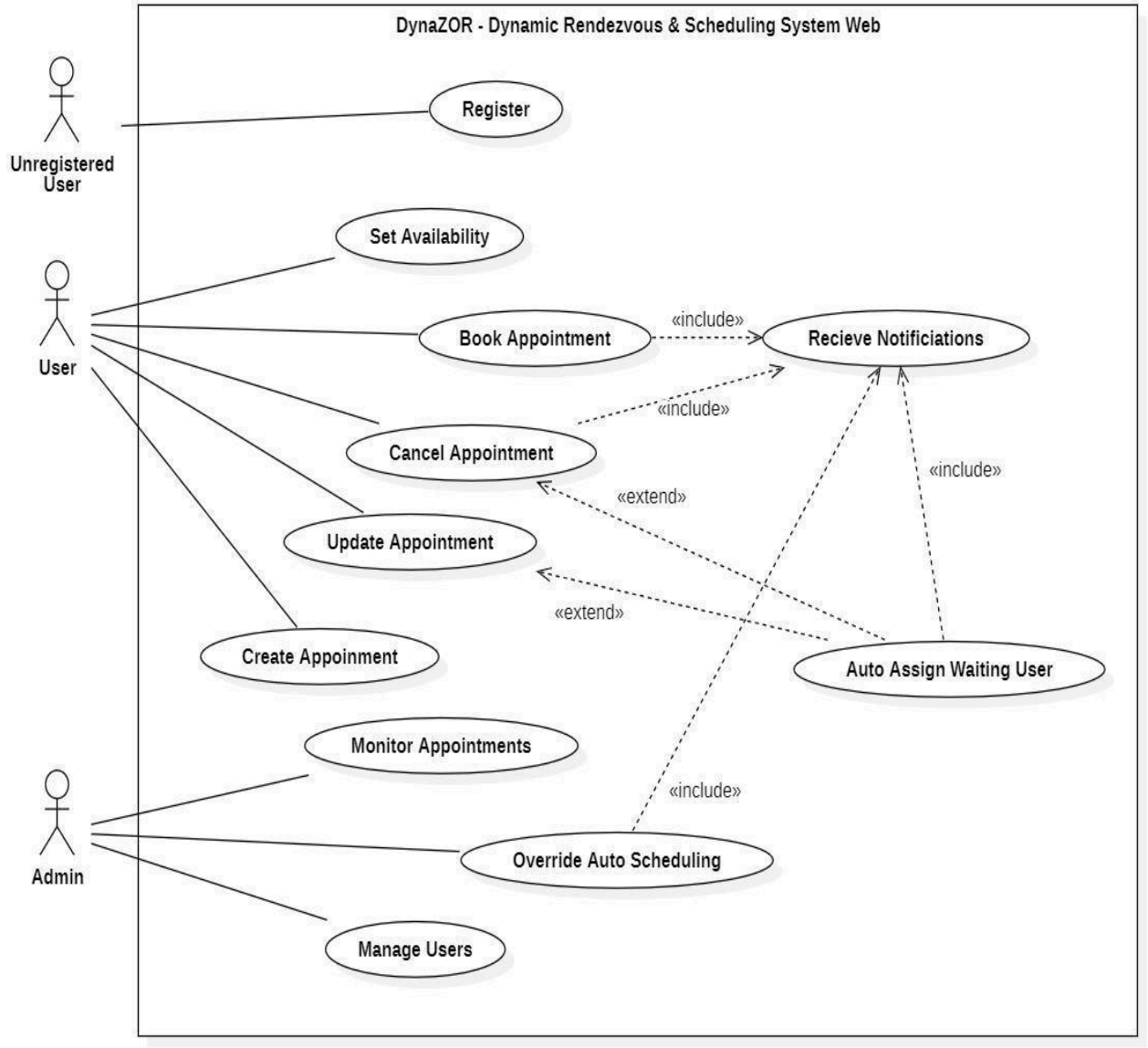

Figure 3: Use case Diagram of DynaZOR

Concerning system functions, the Figure 4 and Figure 5 presenting the DFD diagram level 0 and level 1, explains the overall system functions and necessary input outputs.

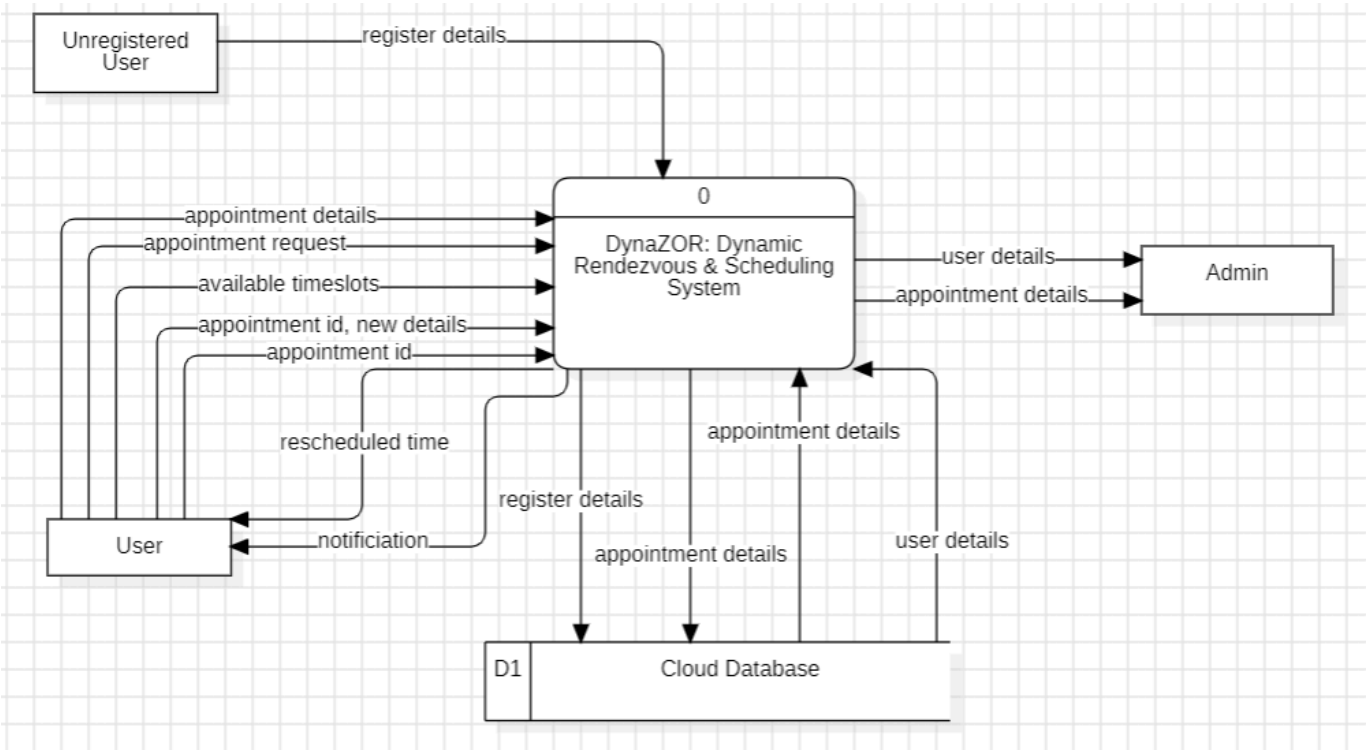

Figure 4: Lvl-0 Data Flow Diagram of DynaZOR

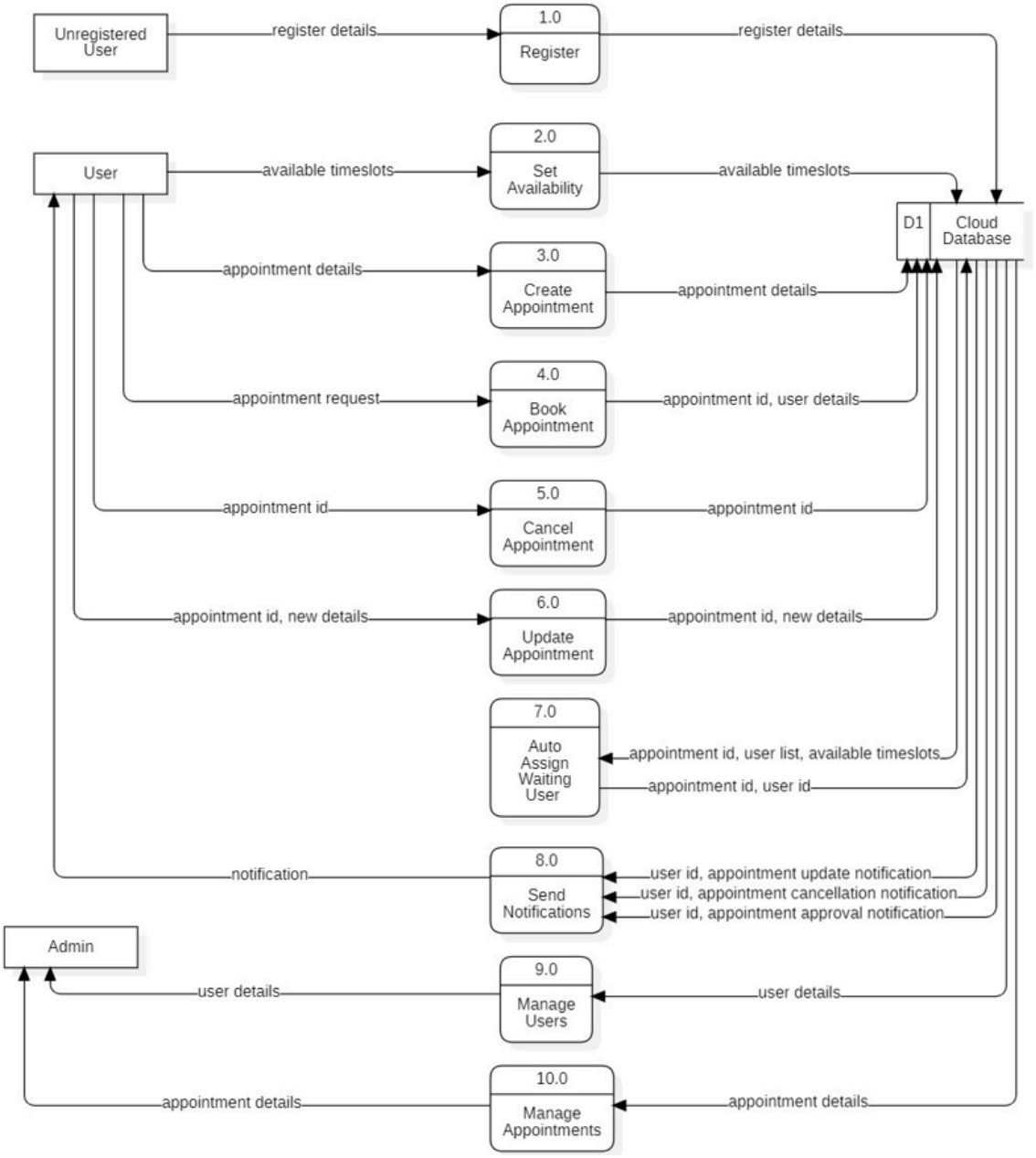

Figure 5: Lvl-1 Data Flow Diagram of DynaZOR

2.3 Technology Stack (Languages, APIs, Frameworks)

2.3.1 Software Development Tools

Frontend Tools

The frontend of the project was developed using Vite as the build tool due to prior familiarity, its rapid hot reload performance, and efficient project setup. Tailwind CSS was incorporated to streamline the styling process. Axios was used to handle communication with the backend through a set of modular API functions. Visual Studio Code served as the primary development environment and was selected because of prior experience with the editor and its overall usability. Together, these tools provided a comfortable and productive workflow for implementing the frontend of the project.

Backend Tools

The backend of the project was developed using Python, more specifically Flask. Flask was used because it is well suited for building RESTful APIs. It has an efficient and simple routing logic. Through Flask, user registrations, logins, database queries and communications are done.Via the pyodbc API, SQL express utilized to create a cloud database structure in the AWS.

Deployment Tools

The project was deployed using GitHub, Vercel, and Render. Render is a Linux based cloud deployment service. On Render's Linux machine, the Microsoft ODBC Driver required to run Microsoft SQL Server was not installed, which made the use of a container manager necessary for deployment. Docker was selected due to its ease of use and good documentation. After successful configuration, Render built the most up to date commit of the selected GitHub repository from the authenticated user and successfully deployed the backend server. Vercel was used to deploy the frontend server; similarly, it also requires a GitHub repository to deploy the server. The frontend server link was added to the CORS configuration so that the backend only accepts API requests from the frontend server and localhost for development. The Vercel deployment environment variable included the backend link so the API responses are configured correctly.

2.3.2 Programming Languages & Frameworks

Frontend Languages & Frameworks

JavaScript serves as the primary programming language with the framework of React. Typescript was also an option but was decided against as not all members had prior familiarity. React was selected as the main frontend framework.

Backend Languages & Frameworks

Python is used as the primary programming language for the backend with the Flask REST framework since it's very efficient, easy to use and mostly used in web developing.

2.4 Cloud Infrastructure & Services Table

2.4.1 Cloud Database Service

Amazon RDS (Relational Database Service) is used as the primary cloud database tool in this project. RDS is a relational database service provided by AWS that helps with backups, scaling and maintenance. It is one of the most important tools for this project, since it will store all the data in its cloud database that is created. Regarding other cloud services & tools used in this project such as Amazon SNS, **Python (PyODBC)** and Microsoft SQL Server, their roles and capabilities and reasoning of these tools being used in this project are written on Table 1.

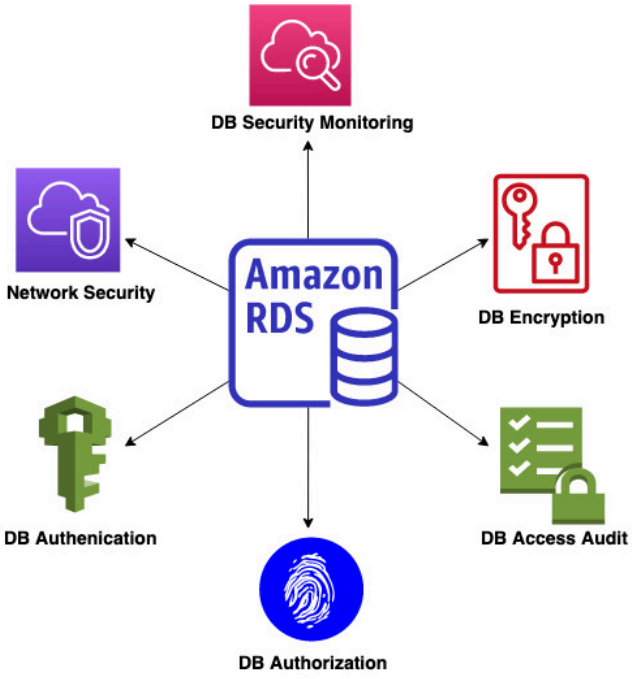

Figure 5: Amazon RDS Abilities

Service Name	Service Model	Role	Justification
Amazon RDS (Relational Database Service)	**PaaS**	**Primary Data Store(and other roles mentioned in Figure 5)**	• **Storage:** Holds all user profiles, schedules, waitlists, and analytics counters. • **Automation:** Handles tasks like automated backups. • **Scalability:** Allows the database to grow as traffic increases.
Amazon SNS (Simple Notification Service)	**PaaS**	**Notification Dispatcher**	**Decoupling:** It is pay-per-request and natively integrates with other AWS services.

SQL Server Express	**Database Engine**	**Local Development**	• **Development:** Used for local testing (as seen in db.py) before deploying to the cloud. • **Compatibility:** Ensures code runs identical T-SQL queries locally and on RDS.
Python (PyODBC)	**Runtime / Connector**	**Backend Logic**	• **Connection:** Acts as the bridge between the application algorithms and the RDS cloud database.

Table 1: Cloud services and tools used in the project.

2.4.2 Cloud Deployment and Hosting Service:

Vercel and Render is used as the deployment and hosting service for this project. Vercel hosts the frontend while Renders hosts the backend of the application. Their communication architecture can be seen in Figure 6 and also Table 2 further explains their roles.

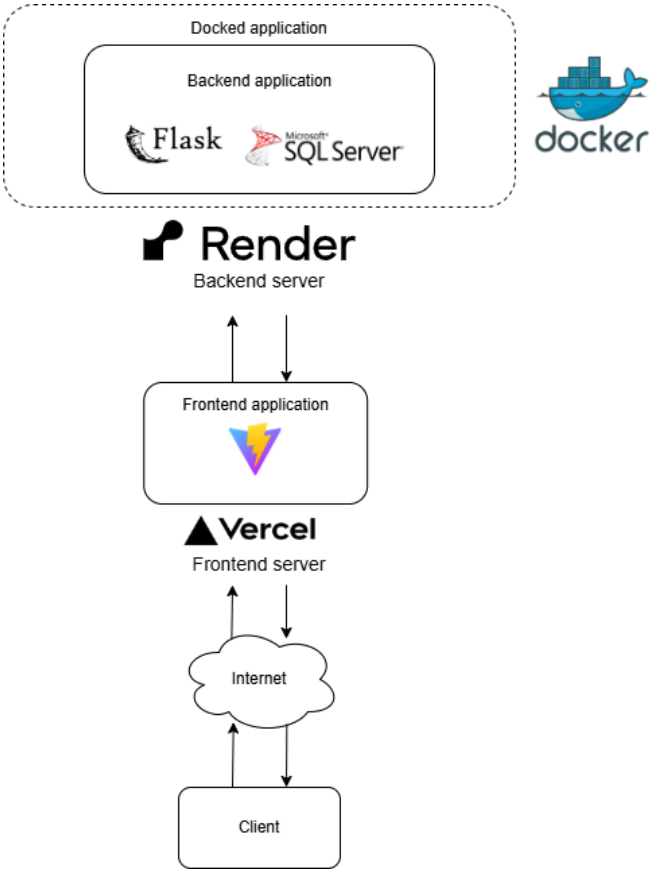

Figure 6: Architecture of hosting service

Service Name	Service Model	Role	Justification
Render	**PaaS**	**Deployment and hosting service for backend.**	• **Deployment:** Builds and runs apps from github repo and additional user set config. • **Automation:** Automatic redeploys. • **Support:** Supports docker based deployments, allowing custom

			runtime configuration and dependency management.
Vercel	**PaaS**	**Deployment and hosting service for frontend.**	• **Deployment:** Builds and runs apps from github repo. • **Automation:** Automatic redeploys. • **Support:** Offers simple environment variable configuration for API base URLs.

Table 2 : Architecture of hosting service

2.5 User Manual & UI Walkthrough

2.5.1 Admin Manual

For enterprise and commercial use, admin pages generally would be deployed conditionally which allows for better protection. These conditions may include VPN only access and IP allowlists. However for a small academic project, these protections are unnecessary and not in the scope of a cloud computing project. So for admin usage, there is only one password authentication to continue. This password is "admin123".

The user has to be routed to the "/admin" page of the frontend deployment link: "https://dyna-zor.vercel.app".

https://dyna-zor.vercel.app/admin:

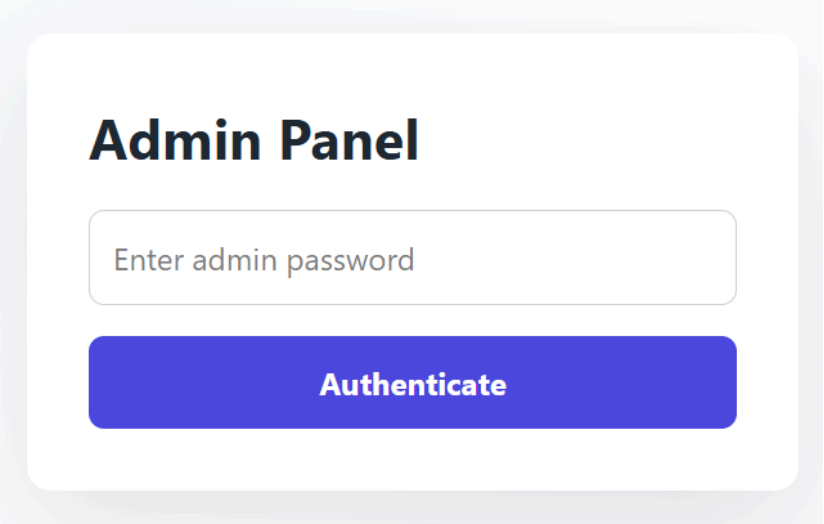

Figure 7: Admin log in

The user will be shown the admin panel after logging from the UI shown in Figure 7. When a user enters the correct password, they are granted entry. Each API call also includes the password with it and the authentication is verified in the backend for all admin api calls.

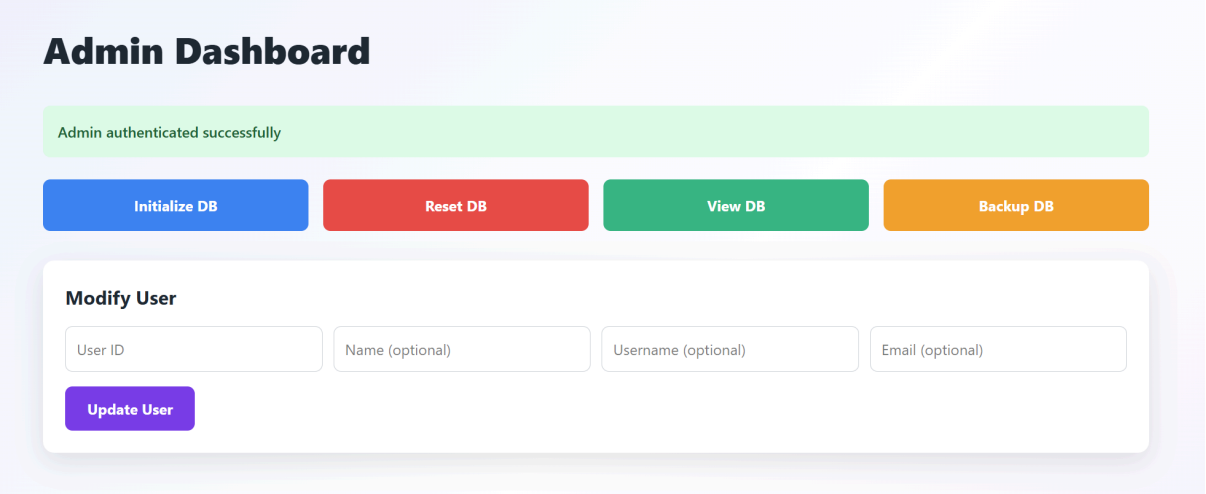

Figure 8: Architecture of hosting service

When authentication is completed the user can see the Admin Dashboard shown in Figure 8. They have the option to initialize, reset, view and backup the database. There is also a panel they can use to modify a user.

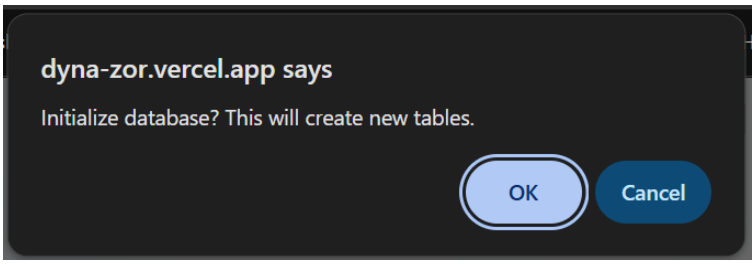

Figure 9: initialize DB button

If the initialize DB button is clicked - Figure 9-, the user will be notified that it will create new tables. This is to minimize the misclicks as initializing the db also resets the tables which is dangerous if done unintentionally.

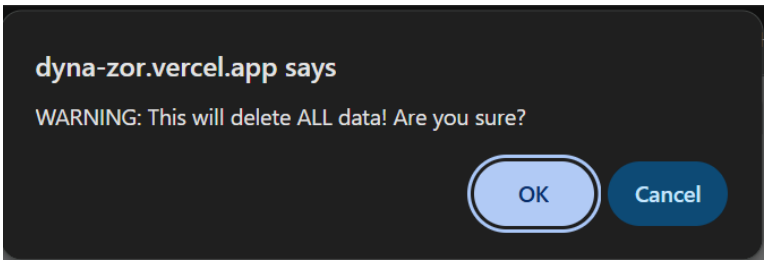

Figure 10: Initialize DB warning

There is warning before proceeding to this action which is shown in Figure 10. A similar warning notification is also shown if the user clicks the reset DB button.

Database Contents

Users (6)

ID	Name	Username	Email	Action
1	Yıldız Alara Köymen	alara124	yildizalarak@gmail.com	Delete
2	test mest	test	test@test	Delete
3	test 2 modified	test_modified	test2@test	Delete
4	c c	c123	c@c.com	Delete
5	d d	d123	d@d.com	Delete
7	Test Testerson	TestGuy	test@test.com	Delete

Figure 11: Delete user buttons

The admin can delete users on the user table via the delete buttons shown in Figure 11.

Schedules (11)

Schedule ID	User	Date
1	alara124 (ID 1)	2025-12-23
2	test (ID 2)	2025-12-23
3	test_modified (ID 3)	2025-12-23
4	c123 (ID 4)	2025-12-23
5	d123 (ID 5)	2025-12-23
7	alara124 (ID 1)	2025-12-24
8	c123 (ID 4)	2025-12-24
9	d123 (ID 5)	2025-12-24
11	test (ID 2)	2025-12-24
12	test_modified (ID 3)	2025-12-24
13	TestGuy (ID 7)	2025-12-24

Figure 12: Users and schedules

Timeslots (153)

Schedule ID
▶ Schedule 1
▶ Schedule 2
▶ Schedule 3
▶ Schedule 4
▶ Schedule 5
▶ Schedule 7
▶ Schedule 8
▶ Schedule 9
▶ Schedule 11
▶ Schedule 12
▶ Schedule 13

Figure 13: Timeslot list

The users schedule, their names and timeslots that belong to the mentioned schedule is listed. Figure 12 and Figure 13 shows this listing. Also, timeslots can be expanded if clicked on a specific schedule as shown in Figure 14.

▼ Schedule 11

Timeslot ID	Time	Available	Booked By
141	08:00	✓ Yes	—
142	08:45	✓ Yes	—
143	09:30	✗ No	—
144	10:15	✓ Yes	—
145	11:00	✓ Yes	—
146	11:45	✓ Yes	—
147	12:30	✓ Yes	—
148	13:15	✓ Yes	1
149	14:00	✗ No	—
150	14:45	✗ No	—
151	15:30	✓ Yes	1
152	16:15	✓ Yes	—
153	17:00	✓ Yes	—
154	17:45	✓ Yes	—

Figure 14: Expanded timeslots

It is important to note here that the booked appointments will continue to be seen as available. Reasoning for that is, other users can also enter a queue because in the cancellation scenario the slot will be reassigned to someone else.

Appointment Stats (34)

Owner ID
▶ Owner ID 1
▶ Owner ID 2
▶ Owner ID 3
▶ Owner ID 4

Figure 15: Appointment statistics table

It can be seen in Figure 15, the appointment stats are listed for Admin. The Admin also can expand this table in case of need for more details which is shown in Figure 16.

Owner ID

▼ Owner ID 1

Booker ID	Time	Bookings
2	10:15	1
2	11:00	1
2	11:45	1
2	12:30	1
2	13:15	1
2	14:00	1
3	11:45	1
7	14:00	1
7	17:00	1
7	17:45	1

Figure 16: Expanded appointment statistics table

Now the user is able to see the statistics for the owner of the schedule.

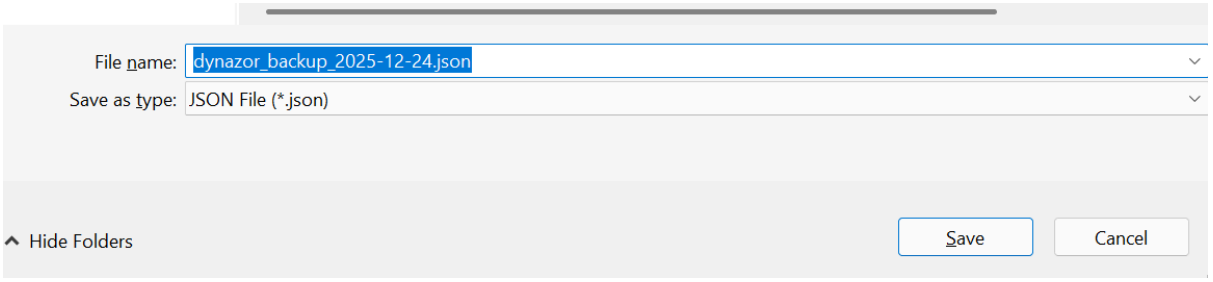

Figure 17: DB backup

When the Admin clicks Backup DB, they can save the JSON file containing the database data as seen in Figure 17.

2.5.1 User Manual

For the client user the UI is fairly straightforward and user friendly.

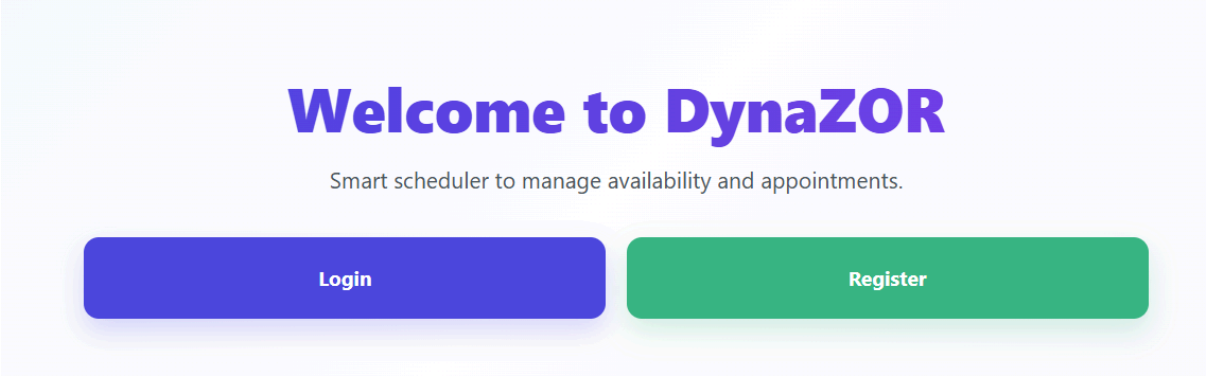

Figure 18: Homepage

Figure 18 shows the homepage the user is greeted by. They have the option to login if they're a returning user, and to register if they're new.

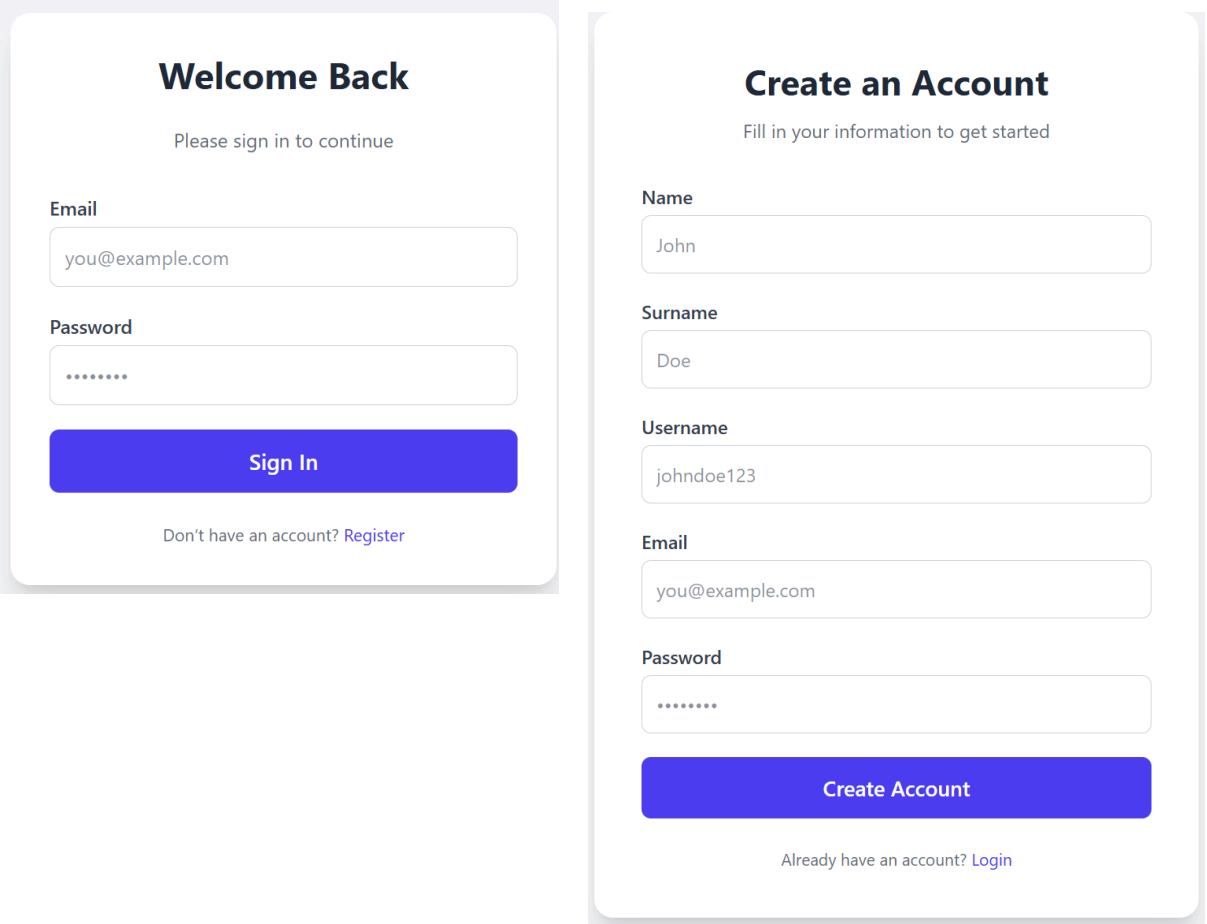

Figure 19: Log in & Sign in

Figure 19 shows the login and the register UI's. They can navigate to the other page by clicking the link at the bottom of both of the components.

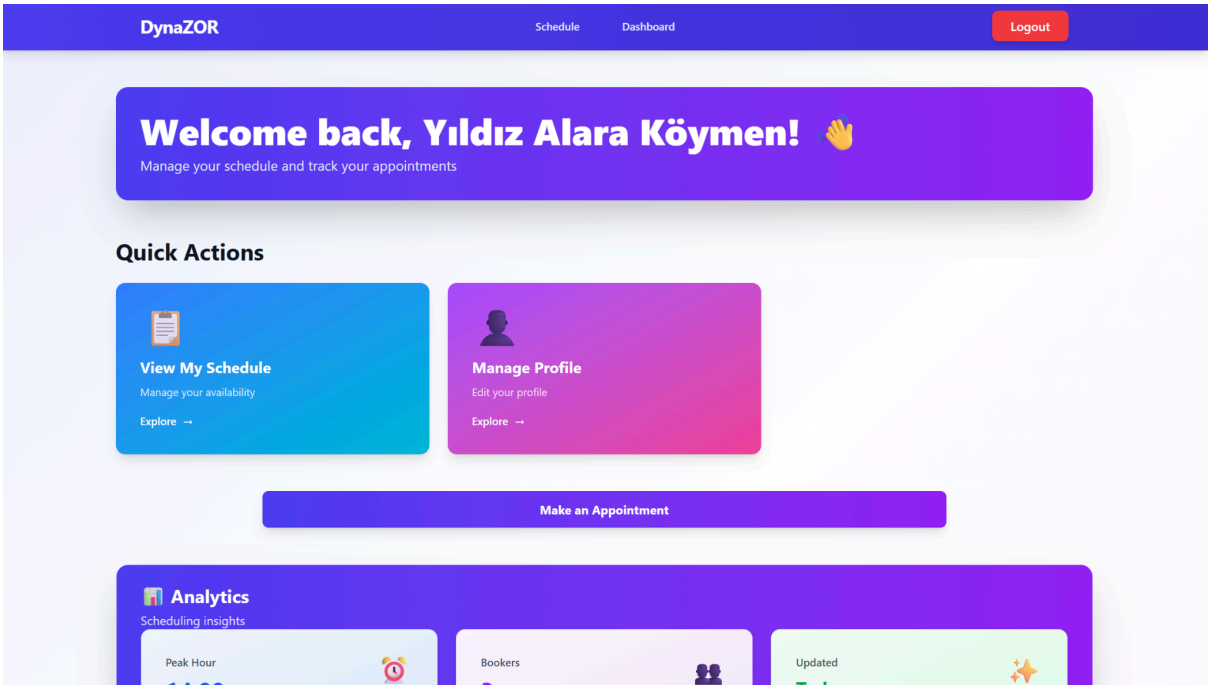

Figure 20: Dashboard

Once the user is authenticated, they are brought to the dashboard shown in Figure 20. Here there is a navbar above that enables users to see their own schedule and come back to the dashboard. They can also logout if they want to.

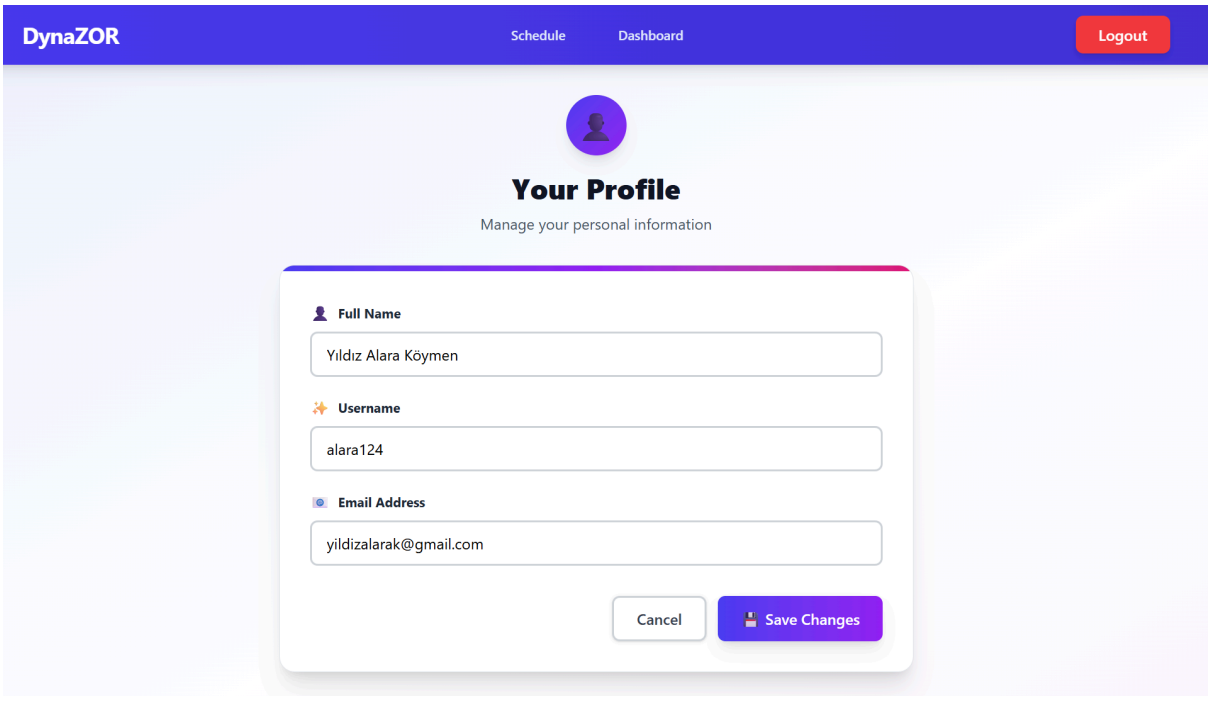

Figure 21: Profile Management

When clicked on the manage profile button, the user is navigated to a panel where they can change their full name, username and email address that can be seen in Figure 21.

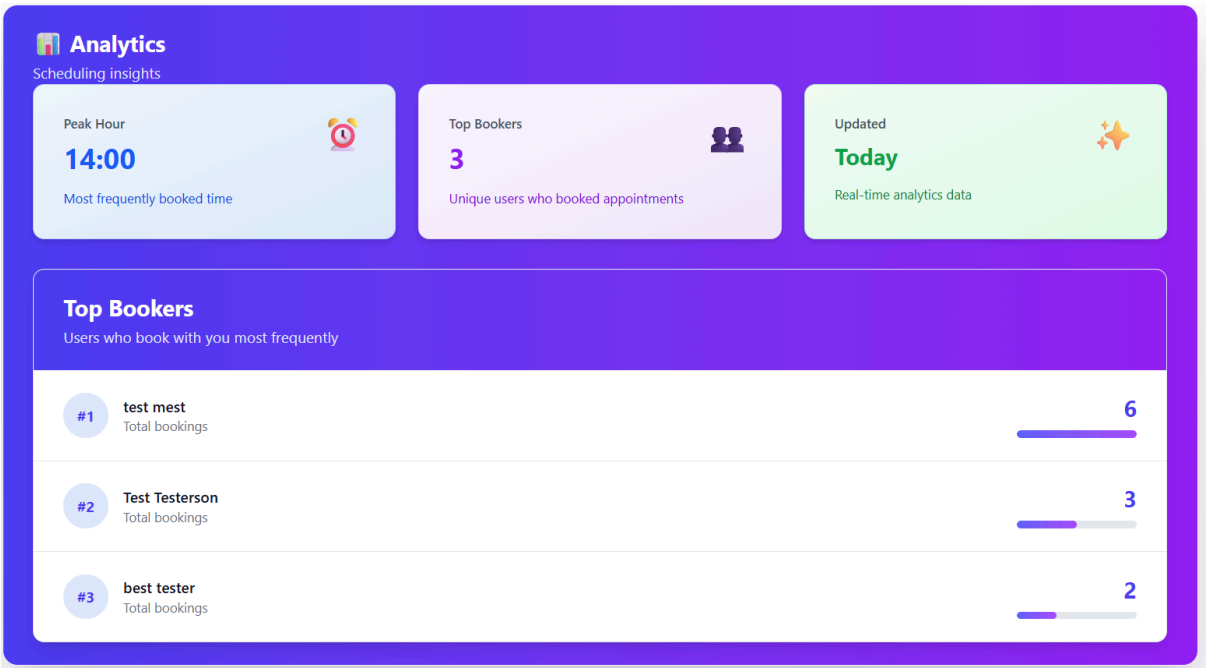

Figure 22: Analytics Dashboard

If the user scrolls down on the dashboard they can see analytics data . In Figure 22, The analytics include which time the users prefer to book and also showcasing the top bookers.

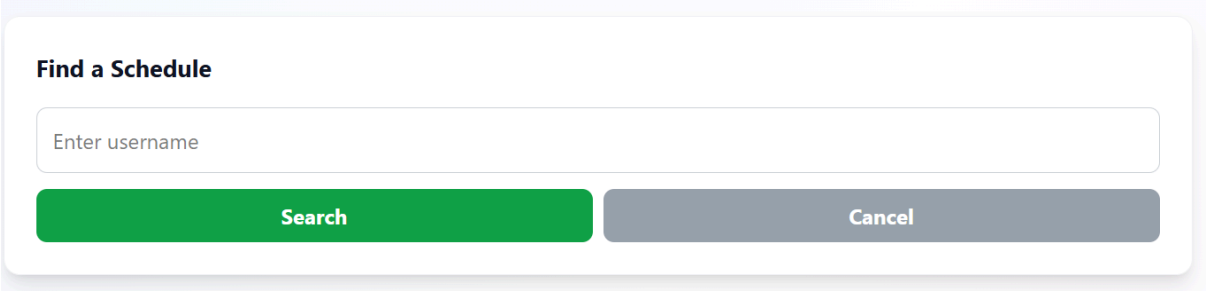

Figure 23: Finding a schedule

If the user clicks on the Make an Appointment button shown in Figure 23, the application prompts the user to enter the username of the person they would like to get an appointment from. The logic of this approach is based on office hours in our school. The people getting these appointments have to be aware of the username of the person they are getting these appointments from. Simply, the person giving out the appointment has to give out this information in real life. For example in academic usage, addressing the username on the syllabus makes sense for this application.

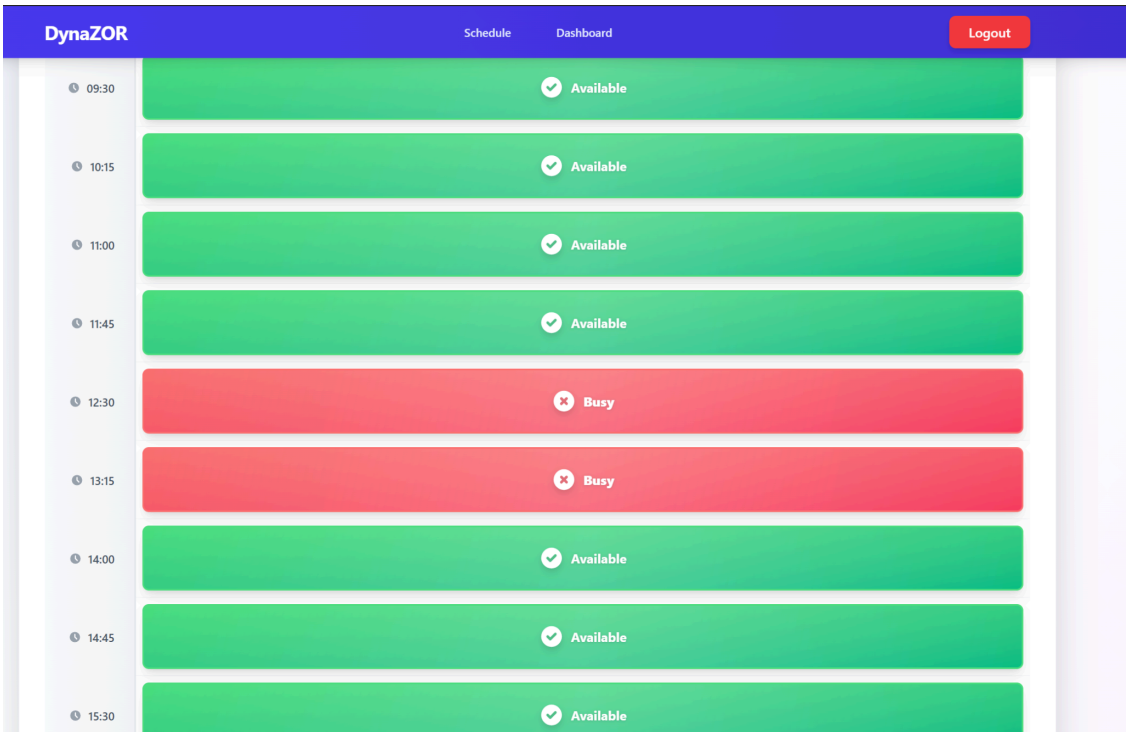

Figure 24: Available and busy cells

When the user enters a username, they are navigated to that user's schedule. The cells shown as “busy” are the cells the owner of the schedule has set as busy themselves, can be seen in Figure 24. If this system can be scaled further in the future, it makes sense for the user to be able to set the interval for their working hours so the system doesn’t have a static showcase of time between 8.45 and 17.45 only. For example the user should be able to set the schedule from 10.15 to 16.15. However, it does make sense to keep the timeslots the same for each user for this application to work smoothly.

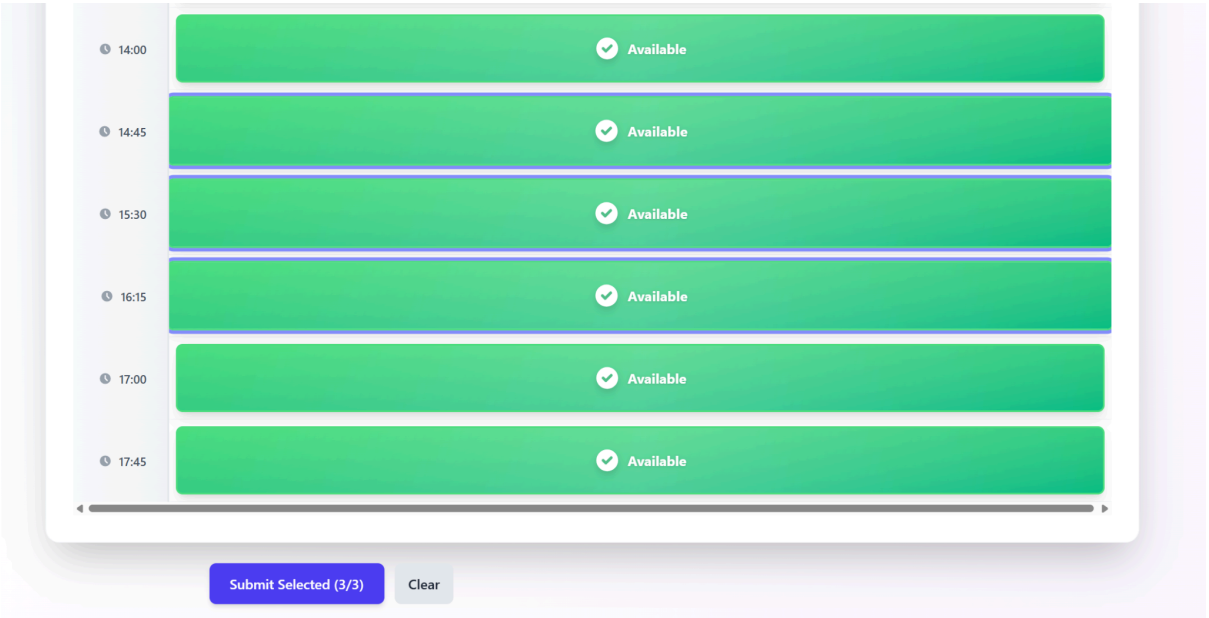

Figure 25: Slot Selection

The user is allowed to get appointments for three timeslots at a time. Figure 25 shows the slot selection and submit selection button. One of the reasons is because as previously mentioned, the inspiration for this scheduler was office hours, realistically there won't be a reason for a user to get more than two slots. However if the need or the usage for this application proves itself to be different, the system is set up for an easy scalability for this case.

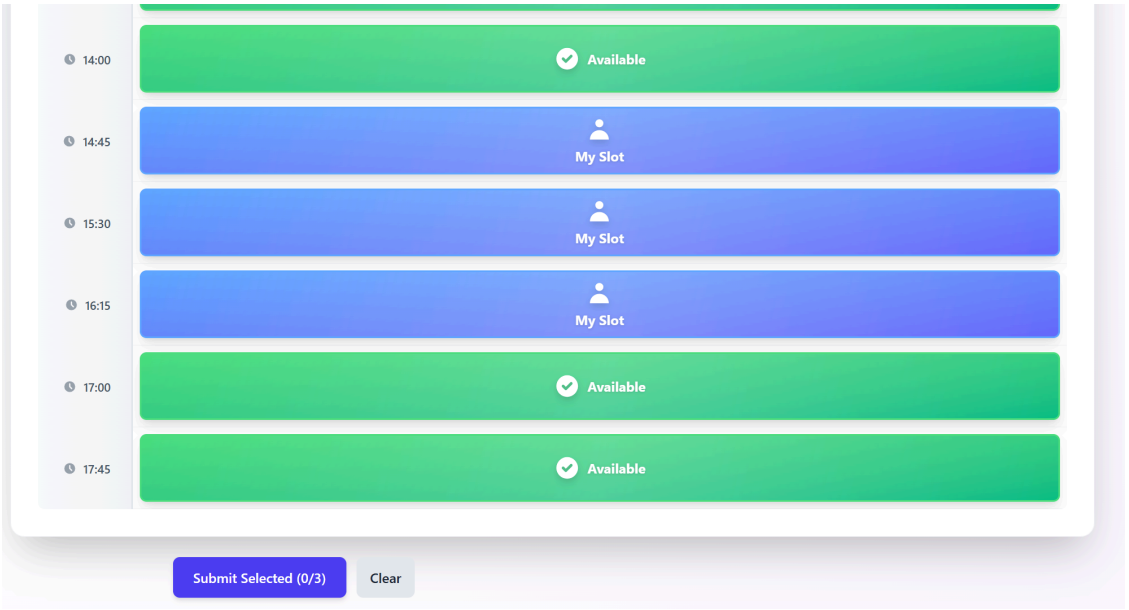

Figure 26: Slot selection after submission

For the user that made the appointment, if there aren't any other users selecting the appointment, the appointment should be given to the user and they can see it as their own slot.

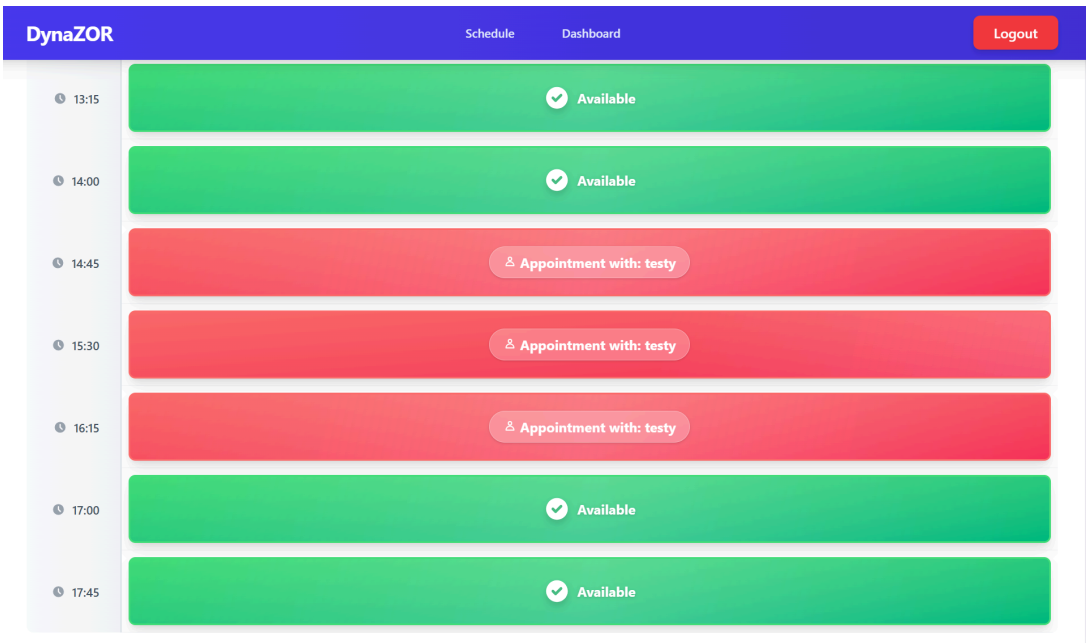

Figure 27: Busy Appointment

If the user navigates back into their own schedule, they can see that their own timeslot is now busy with their appointment shown in Figure 27.

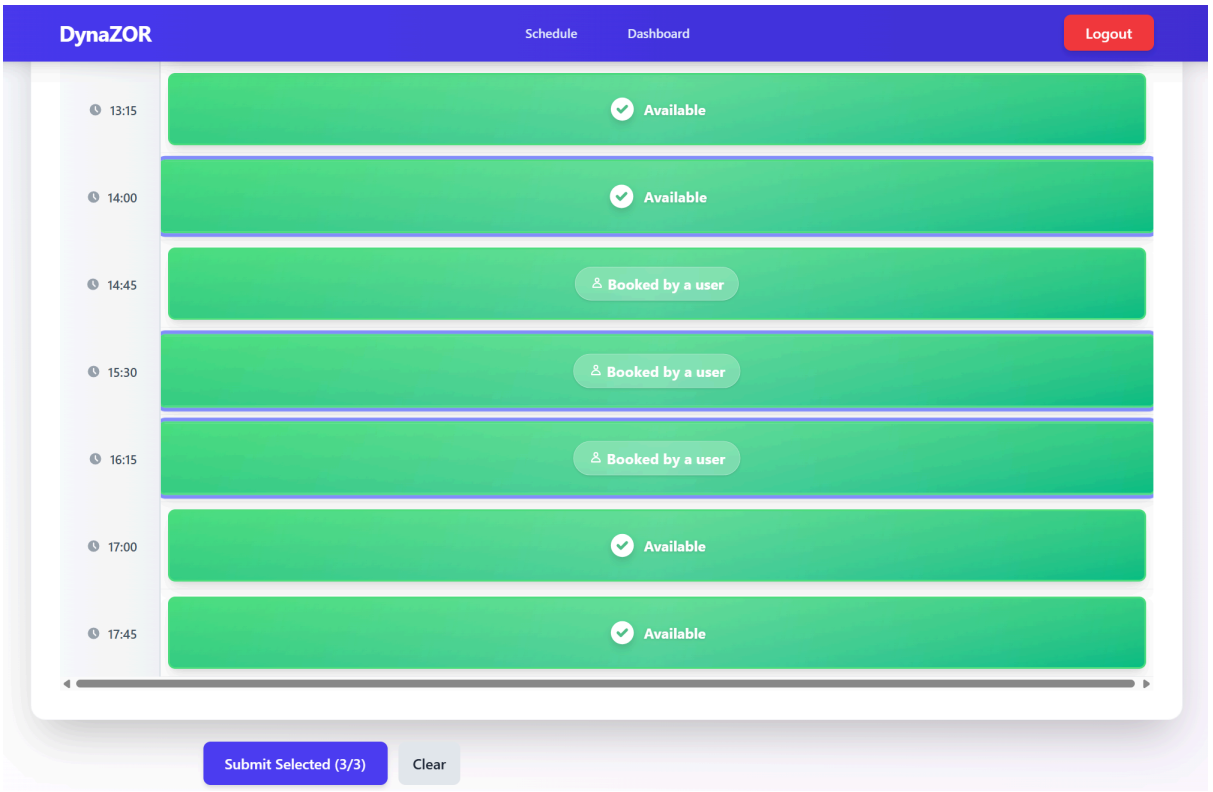

Figure 28: Attempt of User 2

If for example, user 2 attempts to get an appointment from the booked slot user 1 got -they can. This can be seen on Figure 28.

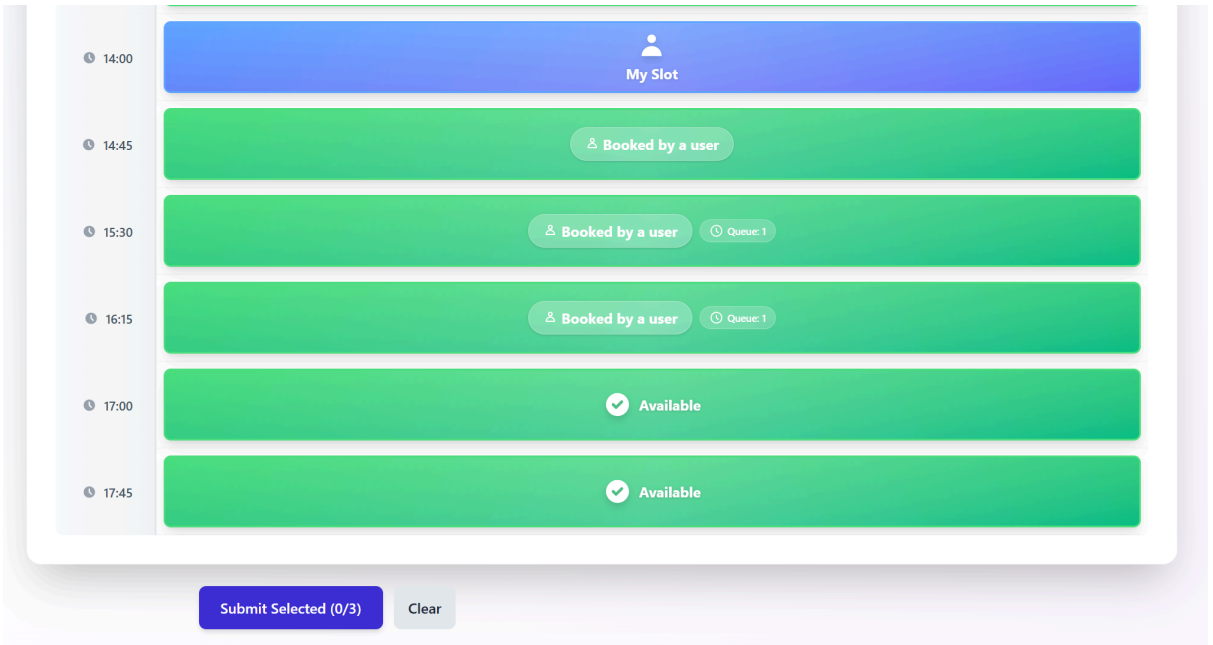

Figure 29: Queue for appointment

However as it can bee seen in Figure 29, the second user wanting the same slot with first user causes a queue to happen which is intended to create rescheduling mentioned before.

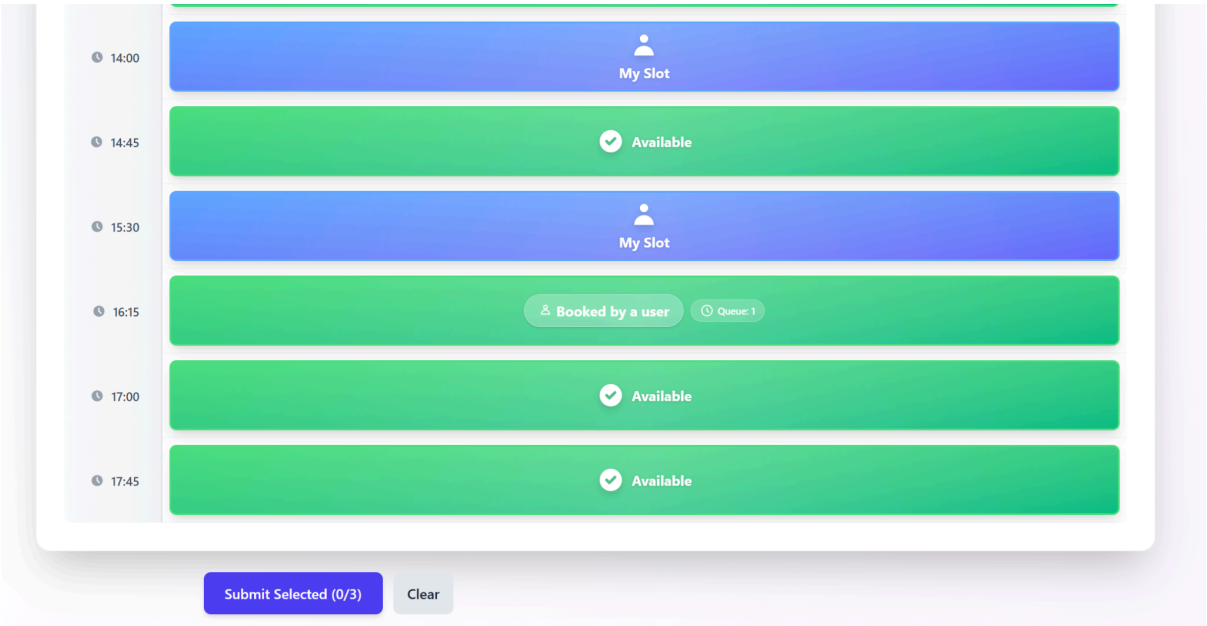

Figure 30: User 1 cancellation.

In this case the first user cancels their appointment for domestic reasons. As it can bee seen in Figure 30, when this event happens, the second user who is in the queue is guaranteed with their choice.

3. Project Statistics

3.1 Development Timeline & Team Responsibilities

Timeframe	**Objective**	**Assigned Team Member**	**Status**
Week 1	• Define project scope & requirements. • Select Tech Stack (Python, SQL, AWS).	Burak Mirac Dumlu Emir Canlı	Completed

		Yıldız Alara Köymen	
Week 2-3-4	• Implement SQL Schema. • Design Entity-Relationship (ER) Diagram. • Enforce Composite Key constraints.	Burak Mirac Dumlu	Completed
	• Setup a local development environment. • Setup and configure cloud database (Amazon RDS) for everyone in the team's use	Emir Canlı	
	• Design the skeleton of basic frontend components. • Setup frontend API calls.	Yıldız Alara Köymen	
Week 4-5	• Scheduling logic. • Implement basic Booking Algorithms. • Create unit tests for backend functions and database utilization creation.	Burak Mirac Dumlu	Completed
	• Backend API endpoints • Backend-frontend integration. • More details on the schedule component. • Login and register api call, database addition tested and working.	Yıldız Alara Köymen	
	• Configuring created algorithms • Integrating created algorithms to backend • Testing and bug fixing	Emir Canlı	

Week 6-7-8	• Implement Priority Queue (Waitlist) logic. •Analytic metrics and database control	Burak Mirac Dumlu	Completed
	• Admin helper api endpoints. • Appointment of frontend id markers. • Setup and implement Amazon SNS system to backend • Testing and bug fixing	Emir Canlı	
	• Dashboard component. • Different schedule render and state transition logic for schedule owners vs viewers. • Schedule cell component revamp with busy and available markers. • Container addition in backend with docker. • Successful deployment. • Admin page. Admin database addition. Admin helper api calls and endpoint bug-fix.	Yıldız Alara Köymen	
Week 9	• Overall system optimization. • Version control system final integration • Finalizing the project & final report.	Yıldız Alara Köymen Emir Canlı Burak Mirac Dumlu	Completed
	• Analytics frontend component. • Manage profile api calls and frontend components. • Addition of analytics table in admin database viewer.	Yıldız Alara Köymen	

	• Backend and frontend testing and bug fixes. • README renditions.	Emir Canlı Yıldız Alara Köymen	

Table 3: Gantt chart of the project

3.2 Key Project Metrics (Lines of Code, Commits, etc.)

While the project started with no metric target, in development some metrics are counted in order to measure performance and impact on cloud servers. The following Table 5 shows the mentioned metrics.

Category	**Metric**	**Value**
Code Structure	Total Lines of Code for Backend.	1,136
	Total Lines of Code for Frontend.	2,225
	Number of Database Tables	5
	Number of API Endpoints	(4-5)
Development	Total GitHub forks	3
	Number of commits	79
	Number of Push/Pull Requests(Sync)	7

	Development Duration	9 Weeks
Performance	Booking Granularity	1 Minute Intervals

Table 5: Table of metrics

4.References

Cal.com. (n.d.). Cal.com: Open scheduling infrastructure. GitHub.
https://github.com/calcom/cal.com

Docker, Inc. (n.d.). Docker documentation.
https://docs.docker.com/

Flask Documentation Team. (n.d.). Flask documentation.
https://flask.palletsprojects.com/en/stable/

Flask-RESTful Contributors. (n.d.). Flask-RESTful documentation.
https://flask-restful.readthedocs.io/en/latest/

GitHub, Inc. (n.d.). Basic writing and formatting syntax.
https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax

Inttter. (n.d.). Markdown badges. GitHub.
https://github.com/inttter/md-badges

Render, Inc. (n.d.). Render documentation.
https://render.com/docs

Tailwind Labs. (n.d.). Installing Tailwind CSS using Vite.
https://tailwindcss.com/docs/installation/using-vite

Vercel, Inc. (n.d.). Vercel documentation.
https://vercel.com/docs

Vite Contributors. (n.d.). Vite documentation.
https://vite.dev/guide/