KUZEY KIBRIS KAMPÜSÜ

ODTÜ METU

NORTHERN CYPRUS CAMPUS

CNG 495

CLOUD COMPUTING

FALL 2025

CAPSTONE PROJECT PROPOSAL

Yıldız Alara Köymen - 2453389

Burak Mirac Dumlu - 2584944

Emir Canlı - 2637544

Table of Contents

1. DynaZOR: Dynamic Rendezvous & Scheduling System

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 storage and notifications. The system dynamically adjusts appointment times when changes occur and instantly notifies users through cloud based alerts, ensuring efficient coordination and improved time management.

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 system also has a strong notification system. When an appointment is scheduled, altered, or cancelled, it provides immediate notifications to all necessary parties. This guarantees that everyone is always up to date, reducing miscommunication and missing meetings. If more information is required for an appointment, such as paperwork or specific details, the system will send a notification with contact information so that everything can be handled without having to send several emails or phone calls.

The system will use the collected appointment data to perform simple analytics and show each user when they are most likely to attend. Using Python pandas library, the system can calculate attendance probabilities for each day and time of the week based on past behavior. For example, analyzing whether a user's schedule is usually full in the morning or weekend appointments. These 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 Tuesdays at 10 AM." This way, users can adjust their future habits and create appointment times that fit their routines more reliably.

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. The project will be implemented as a Software as a Service (SaaS) solution. Users will be able to access the system through a web interface without needing to install or manage any local software.

2. Implementation

The implementation of DynaZOR combines backend and frontend components to form a functional scheduling system. The backend manages data and storage, while the frontend, built with React, handles user interaction and display. AWS S3 provides cloud-based data storage. Using Flask for the backend and React for the frontend as separate servers ensures a clear separation of concerns and simplifies development.

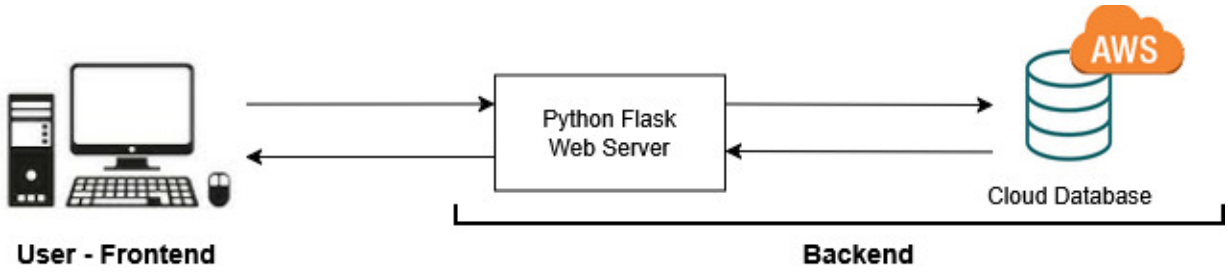

Figure 1: System architecture of DynaZOR

2.1 Backend

The backend of DynaZOR will be developed using Flask. The team is already experienced with Python and Flask, which makes it a suitable and efficient choice for implementing the core application logic. Flask will manage appointment handling, user information, and the dynamic rescheduling operations of the system.

The backend will implement RESTful endpoints to handle client requests and exchange data in JSON format. After processing, it will use the AWS SDK (Boto3) to interact with Amazon S3 for data storage. AWS S3 (Simple Storage Service) is a scalable and reliable cloud storage service that allows storing and retrieving data objects in the form of files. In the DynaZOR system, AWS S3 will be used to store structured data such as JSON files containing appointment records, user availability, and system logs.

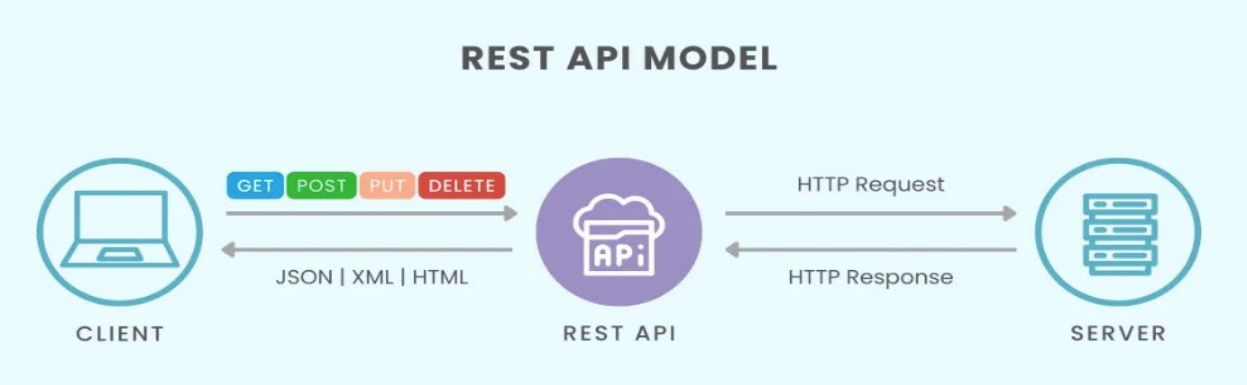

Figure 2: Architecture of REST API model (Yu, 2023)

Flask will use Boto3 to read, write, and update data objects within S3 buckets. Each operation, such as adding or updating an appointment, will correspond to creating or modifying a file in S3. This setup provides a scalable backend structure suitable for the scope of this project.

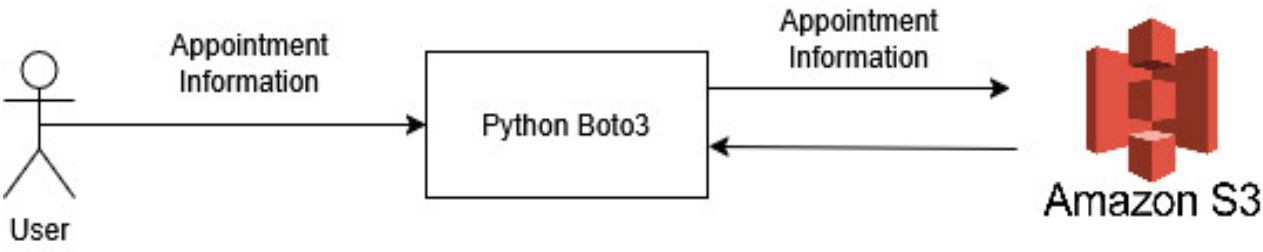

Figure 3: Data flow between the user, Python Boto3, and Amazon S3

2.2 Frontend

The frontend of DynaZOR will be developed using React. React is chosen because it allows the reuse of components, which helps in building features like calendars, time-slot grids, and appointment lists efficiently. The team is already familiar with JavaScript, so using React simplifies the development process and reduces the learning curve.

React's component-based structure also makes it easier to maintain a clear and organized codebase. It supports efficient updates when appointment data changes, which is important for DynaZOR's dynamic scheduling and rescheduling features.

2.3 Cloud Notification System

Amazon Simple Notification Service (SNS) will be integrated into the Flask backend to handle user notifications for rescheduled and cancelled appointments. Whenever an appointment is modified or cancelled, Flask will trigger an SNS publish event that sends a notification to the corresponding users. This ensures that users are promptly informed about any changes in their schedules without manually checking the system. Because this approach is scalable, we are allowed to include other notifications based on the development evolution of the system. SNS topics will be configured to deliver messages through email. AWS SDK (Boto 3) will be used here to access Amazon SNS services.

3. Diagrams

3.1 Data-Flow Diagram:

The data flow diagram illustrates how information moves through the DynaZOR system. Its purpose is to give a clear overview of how data is processed and transferred within the system.

3.1.1 Level-0:

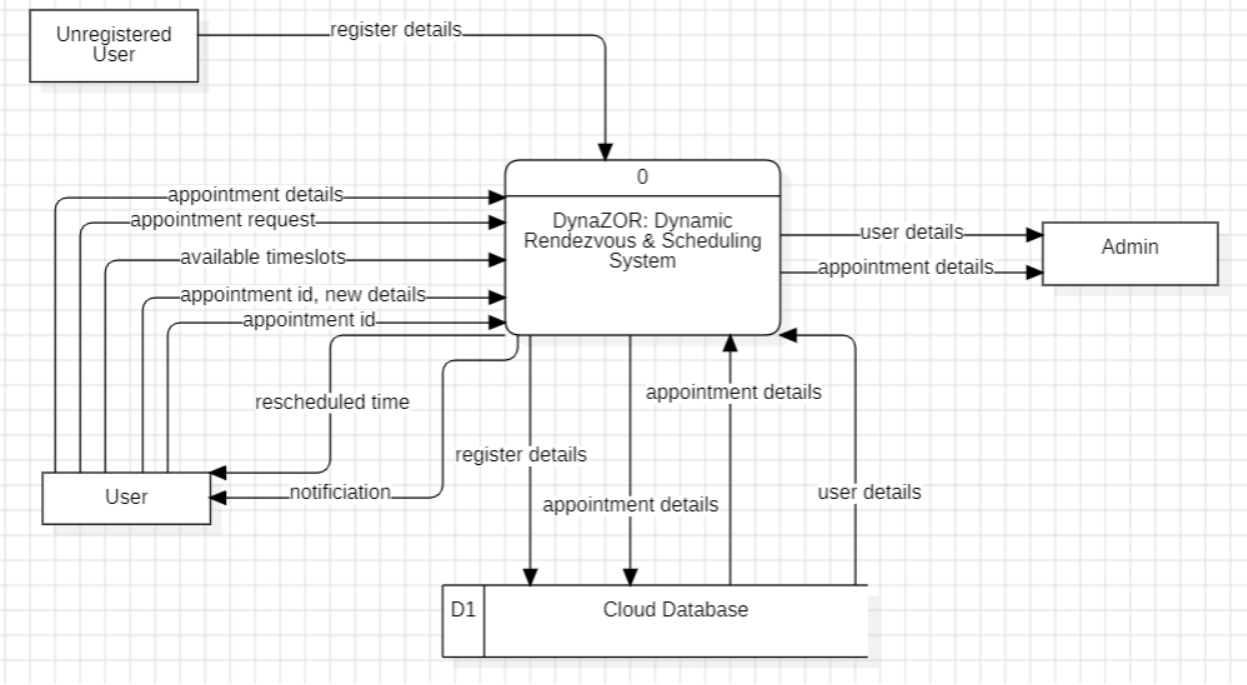

Figure 4: Level-0 Data-Flow Diagram of DynaZOR

3.1.2 Level-1:

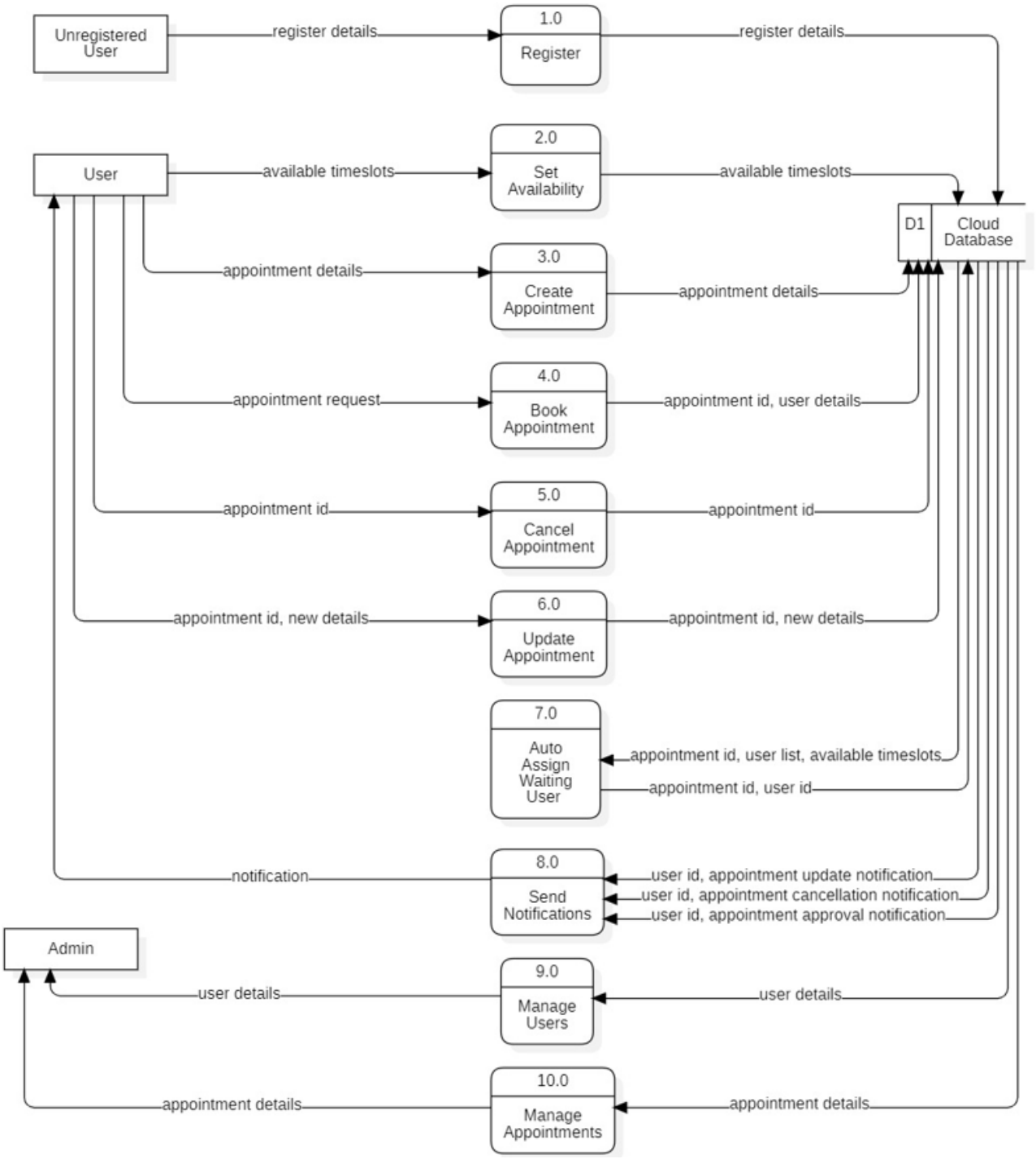

Figure 5: Level-1 Data-Flow Diagram of DynaZOR

3.2 Use case Diagram:

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.

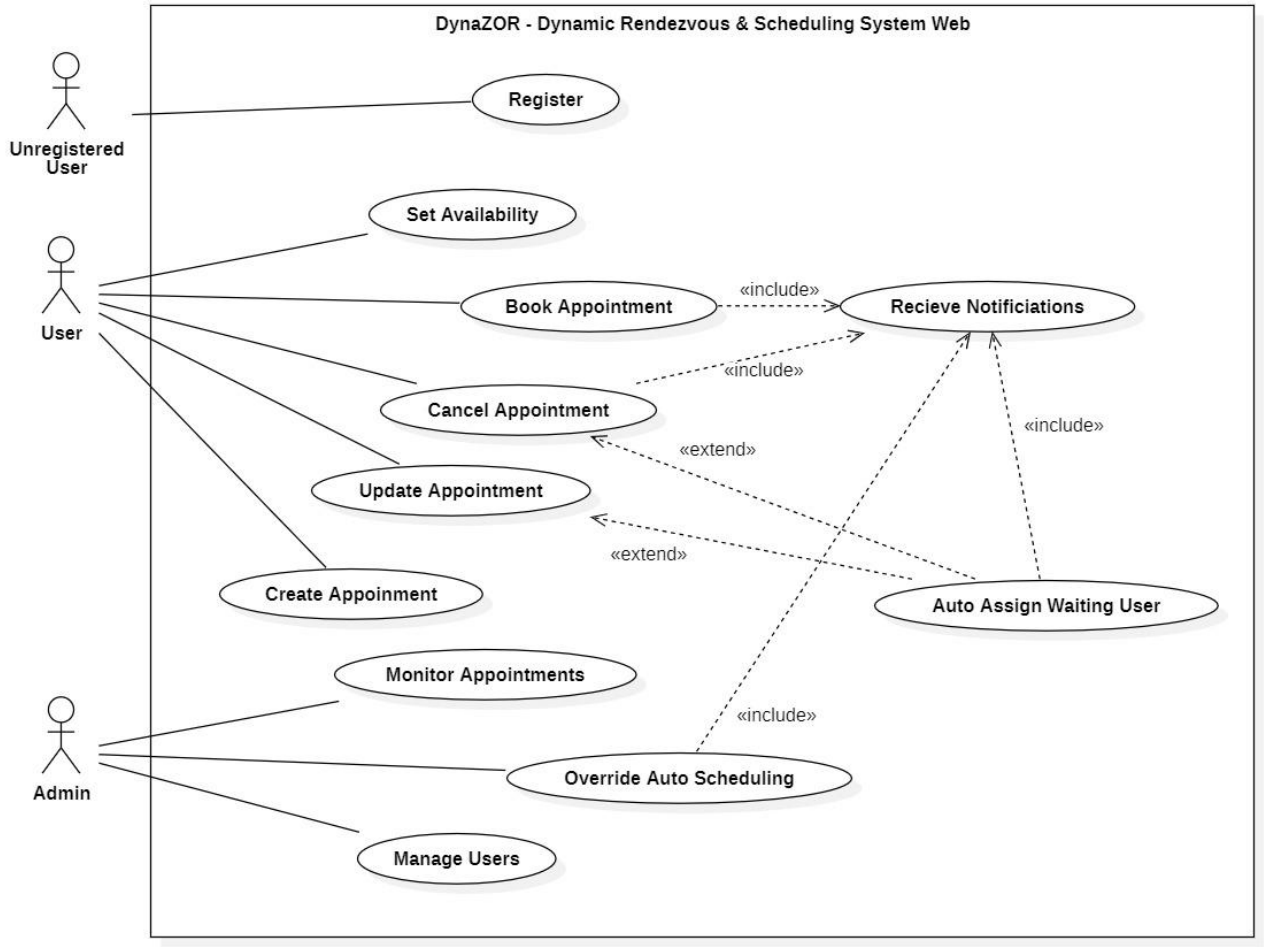

Figure 6: Use case Diagram of DynaZOR

4. Data Types

The system primarily handles **JSON data** for user and appointment information.

Examples:

- appointment.json: {user_id, provider_id, date, time, location}
- users.json: {name, email, preferred_hours}

5. Computation

The backend performs key computations related to scheduling optimization and conflict detection. When a new appointment is created or an existing one is rescheduled, the system calculates available time slots based on all users' preferences and existing bookings. The backend handles these computations before updating the cloud storage.

6. Expected Contribution

- **Yıldız Alara Köymen:** Frontend development with React, Flask API design.
- **Burak Mirac Dumlu:** Backend AWS database integration.
- **Emir Canlı:** Backend SNS integration, cloud setup.

7. References

Amazon Simple Notification Service Examples. (n.d.). Retrieved from AWS SDK for JavaScript: https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sns-examples.html

Boto3 documentation. (n.d.). Retrieved from amazon aws: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html

Flask Documentation. (n.d.). Retrieved from Flask: https://flask.palletsprojects.com/en/stable/
React. (n.d.). Retrieved from React: https://react.dev/

Yu, I. (2023, October 31). *[Part 2] REST API components & How to read them*. Retrieved from SkipLevel: https://www.skiplevel.co/blog/part-2-rest-api-components-how-to-read-them