



**ORTA DOĞU TEKNİK ÜNİVERSİTESİ**  
**MIDDLE EAST TECHNICAL UNIVERSITY**  
KUZEY KIBRIS KAMPUSU ♦ NORTHERN CYPRUS CAMPUS

**CNG 495**  
**CLOUD COMPUTING**  
**METU NCC Society Website**

**FALL 2024 – Final Report**

Team Members	
Ekrem Cagatay Goz	2526374
Haya Arabi Katibi	2542520
Engin Eray Kabalak	2526424

## Table of Contents

Table of Figures .....	4
Introduction .....	5
Benefits .....	5
Novelties .....	5
Similar Projects .....	5
GitHub Repositories .....	5
Client Side.....	5
Server Side.....	5
Structure of the project .....	6
Front-End.....	6
1. Header/Footer Components .....	6
2. Login/Signup Page .....	6
3. Dashboard.....	6
4. Society Page.....	6
5. Reservation Requests .....	7
6. Add Society .....	7
7. Booking Form.....	7
8. Edit Event.....	7
9. My Societies.....	8
Backend .....	9
1. User Authentication.....	9
2. User API .....	9
3. Societies API.....	9
4. Announcements API .....	9
5. Join Request API .....	9
7. Reservation Request API .....	10
8. S3 Services Used in Announcement API'S.....	10
Cloud Integration .....	11

User Manual.....	12
Main Dashboard .....	12
Login Screen .....	13
Register Screen.....	13
Admin Operations .....	14
Admins Dashboard .....	14
Admin View Society .....	14
Reservation Request.....	15
Add Society .....	15
President Operations .....	16
President Dashboard .....	16
Add Announcements .....	17
Show Reservations.....	18
Requests .....	18
User Operations .....	19
User Dashboard .....	19
Diagrams .....	20
List of Technologies.....	22
Explanation and Difficulties .....	22
Project Statistics.....	24
Time Frame.....	24
Number of Lines Code.....	25
Memory Requirements.....	26
Database .....	27
References .....	28

# Table of Figures

Figure 1 cloud services utilized .....	11
Figure 2 Main Dashboard .....	12
Figure 3 Login Screen.....	13
Figure 4 Sign up Screen.....	13
Figure 5 Admins Dashboard .....	14
Figure 6 Admins View Society Screen .....	14
Figure 7 Reservation Request Screen .....	15
Figure 8 Add Society .....	15
Figure 9 President Dashboard .....	16
Figure 10 Society screen for president .....	16
Figure 11 Add Announcements .....	17
Figure 12 Room Reservation .....	17
Figure 13 Show Reservations .....	18
Figure 14 Join Requests Screen .....	18
Figure 15 User Dashboard .....	19
Figure 16 User Joined Societies Screen.....	19
Figure 17 Database Configuration .....	27
Figure 18 Database Storage .....	27

# Introduction

METU Society Application is developed to manage the societies, events, announcements and room reservation in a modern and more efficient fashion. Societies will be able to share their announcements through the webpage and update them according to their needs. Society members will have access to the events held by societies and their details such as date, time, location etc.

## Benefits

This app simplifies and makes it more convenient for both society president and members to follow up to recent events. Allowing a smoother dynamic between them. Moreover, societies will be able to reserve a location in CCC in few clicks without having conflicts with other societies events, making the process organized for all societies.

## Novelties

Our novel approach focuses on creating a real time interactive website designed specifically for universities. Our project enhances the student experience in campus life by making it more active, social and enjoyable.

## Similar Projects

Other platforms similar to ours are available such as Google Calendar and Slack. However, those platforms are not designed specifically for universities and lacks the role-based access control.

## GitHub Repositories

### Client Side

<https://github.com/cagatay-goz/society-app-client.git>

In this repository, we pushed our front-end side. For front-end, we used React.js technology with JavaScript since its implementation is easier than pure HTML and CSS.

### Server Side

<https://github.com/cagatay-goz/society-app-api.git>

This repository consists of our back-end side. For this side, we used Spring Boot technology with Java language since its deployment duration is faster than the other frameworks.

# Structure of the project

## Front-End

### 1. Header/Footer Components

These components are essential for the user navigation experience. They provide links or buttons that help users navigate to different parts of the application, according to their role (e.g., admin, society president, or regular member).

Header: Contains navigation links such as "Home," "Login/Signup," "Dashboard," and "Logout" (for logged-in users) etc. It is designed to be visible on all pages for easy access.

Footer: Includes the copyright information

### 2. Login/Signup Page

The login/signup page handles the user authentication process. It ensures only authorized users can access the app based on their credentials.

Login Form: The user is asked to input their email and password. Upon submission, the frontend sends a request to the backend to validate the credentials.

Signup Form: A new user can create an account by providing necessary details such as email, and password. This data is sent to the backend, where it is stored in the database for future logins.

### 3. Dashboard

The dashboard serves as the central place for users to explore and interact with the societies available in the system.

Displays a list of societies that the user can view and interact with. Each society will be represented with key information like its name, description, and a link to the specific Society Page.

The dashboard provides an overview of the societies and updates dynamically, allowing users to join or follow new societies.

Interactive elements: Users can click on view button to be redirected to its detailed page.

### 4. Society Page

The Society Page provides detailed information about a particular society, including its announcements, activities, and media.

Society Info: Displays the society's name and description. Announcements: Shows the latest posts or events from the society, including images and descriptions.

The page provides deeper engagement with the society's content for users interested in joining or participating.

## **5. Reservation Requests**

This component manages the requests for event location bookings, visible to admins who have control over approving or rejecting the requests.

Pending Requests: Displays a list of pending reservation requests made by society presidents to book a venue for their events.

Admin Control: The admin can either accept or reject the request, allowing for better management of space allocation and preventing double bookings.

## **6. Add Society**

Allows the admin to add new societies to the system. This helps expand the app's content and makes it possible for new societies to be managed through the platform.

Add Form: The admin fills out a form containing fields such as the society's name, description, president's details, and other necessary data.

Submission: Once the form is completed, the admin submits it, and the new society is added to the system's database, making it visible to users in the dashboard.

## **7. Booking Form**

This form enables society presidents to request a location and time for their event.

Event Details: The form asks for details such as the event name, description, preferred time and date.

Location: It allows the president to choose a location (e.g., a room in the campus).

Once submitted, the request is sent to the backend, which handles the event reservation process.

## **8. Edit Event**

This page allows society presidents to modify an existing event they have created, making it easier to update details.

The form will pre-populate with the existing event details (like event name, date, time, description). The president can make changes and submit them.

Backend Interaction: The updated information is sent to the backend, which modifies the event record in the database.

This functionality ensures that the event details stay accurate and up-to-date.

## **9. My Societies**

This page allows users (especially regular members and presidents) to see a list of societies they are a member of.

Society List: Shows a list of societies that the logged-in user is a part of, with details like the society's name and its activities.

Interactivity: Users can click on any society to view its detailed page, or they may leave the society from this page.

This page helps users quickly access societies they are involved in and stay updated on their activities.



# Backend

## 1. User Authentication

We have implemented Signup and Login APIs to handle user authentication. These APIs ensure secure user management and authentication processes:

## 2. User API

1. Allows a new user to register with their details. (Returns: A success message or error response.)
2. Authenticates a user and returns a JWT token upon successful login. (Returns: A JWT token and user email.)

## 3. Societies API

1. Get All Societies and Their Information
2. Get Specific Society Information
3. Add New Society
4. Get Societies for Specific User (Helps the user to see their enrolled societies.)

## 4. Announcements API

1. Retrieves a list of all announcements. (Returns: A list of all announcement objects.)
2. Retrieves a specific announcement by its ID. (Returns: A single announcement object corresponding to the given ID.)
3. Retrieves announcements belonging to a specific society by society ID. (Returns: A list of announcement objects related to the specified society.)
4. Creates a new announcement using multipart form-data. (Returns: The created announcement object.)
5. Deletes an announcement by its ID. (Returns: A success message or status code.)
6. Updates an existing announcement using the provided UpdateAnnouncementRequest. (Returns: The updated announcement object.)

## 5. Join Request API

1. Creates a join request for a user to join a specific society using their email and the society ID. (Returns: A success message or an error message if the user is not found.)
2. Processes a join request by its ID with a specified action (approve or reject). (Returns: A success or failure message based on the action performed.)
3. Retrieves all pending join requests for a user based on their email. (Returns: A list of pending join request details.)

## **7. Reservation Request API**

1. Creates a new reservation request using the provided details. (Returns: A success message upon creation.)
2. Retrieves all reservation requests visible to a specific admin based on their email. (Returns: A list of reservation request details.)
3. Processes a reservation request by its ID with a specified action (accept or reject). (Returns: A success or failure message based on the action performed.)
4. Retrieves all reservation requests created by a specific president based on their email. (Returns: A list of reservation request details for the president's society.)

## **8. S3 Services Used In Announcement API'S**

1. Uploads a file to an S3 bucket and returns its URL. (Returns: The URL of the uploaded file.)
2. Deletes a file from the S3 bucket using its URL. (Returns: A success or failure message based on the operation.)

## Cloud Integration

**AWS RDS:** Manages the relational database that stores user data, society details, and announcements. Functions include data storage, retrieval, and security management.

**AWS S3:** Stores images associated with society announcements. Provides unique URLs for each image to be used in the frontend.

**AWS IAM:** Manages user access and permissions for AWS resources. This ensures that only authorized users can access certain resources.

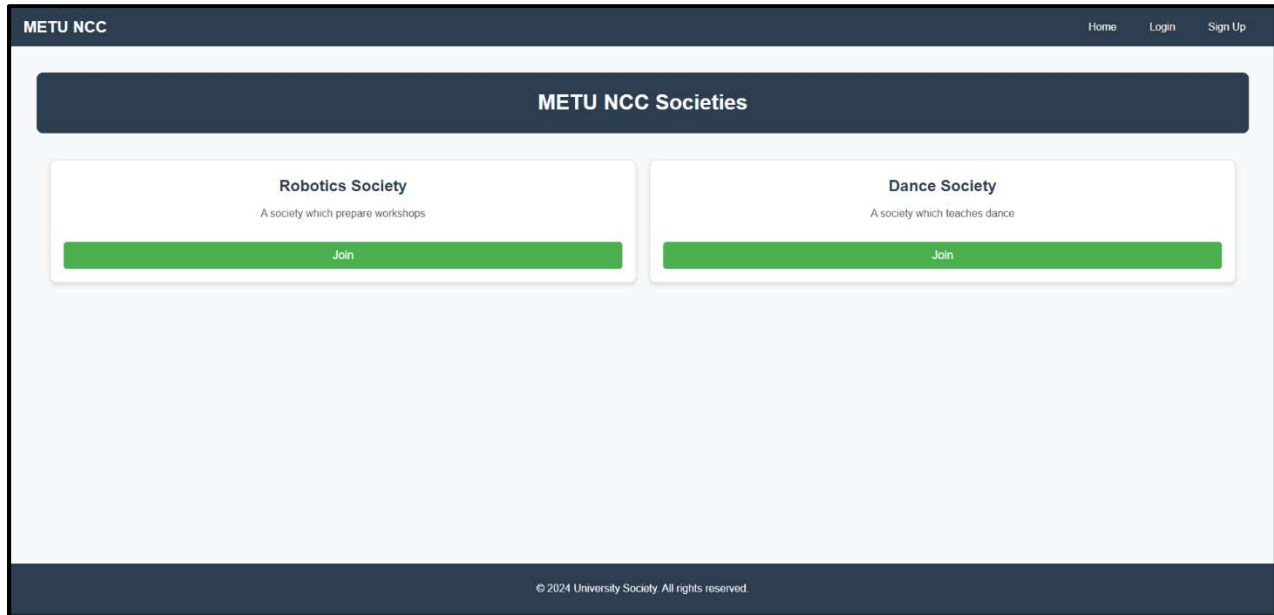
**AWS EC2:** Manages the virtual servers (instances) that host the backend and frontend of the application. These instances provide the computational power to run the web application, process requests, and communicate with other AWS services like RDS, S3, and IAM. AWS EC2 ensures scalability, allowing the application to handle increased user traffic when necessary.

SERVICES	FRONTEND	BACKEND	AWS PORTAL
AWS RDS		Database creation and usage at the backend.	See Database and Neccessary Informations.
AWS S3	Using Url show announcement image on the screen	Store Announcement Image to bucket	See Bucket details and files in the bucket.
AWS IAM			Give Necessary Permissions to 1-am users for using AWS services
AWS EC2	Deployed front-end using Ubuntu server with NGNIX	Run back-end build on local of server, and front-end sends these requests to this local	Gave the server computer, EC2, to run our two sides

*Figure 1 cloud services utilized*

# User Manual

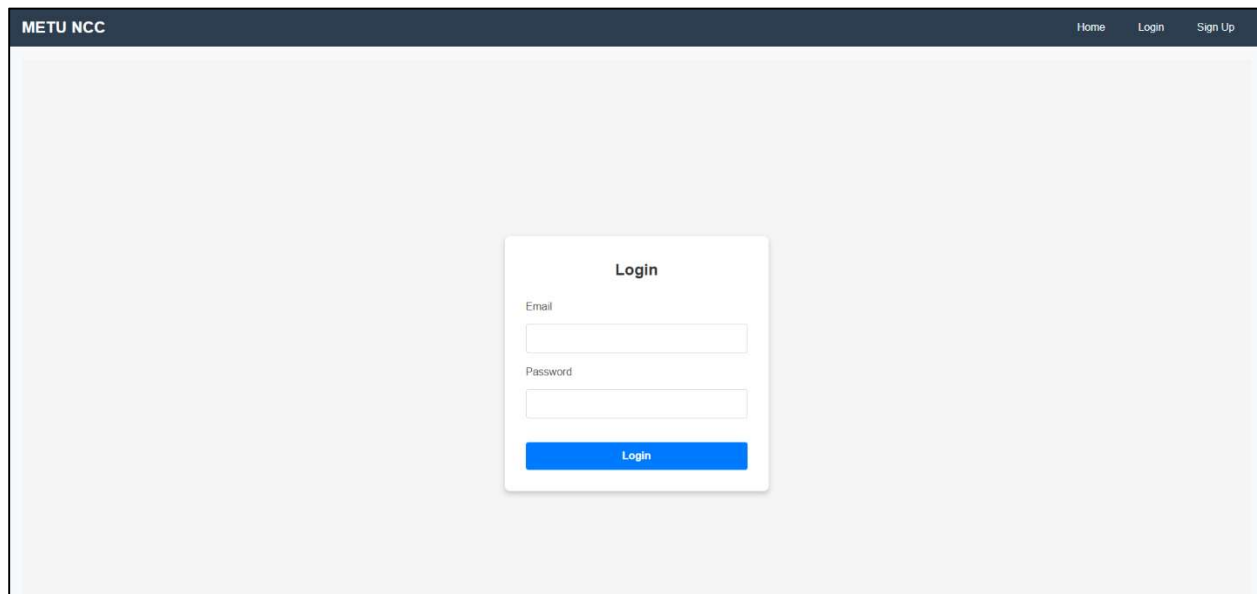
## Main Dashboard



*Figure 2 Main Dashboard*

The first screen visible to user whether logged in or not is the dashboard. Dashboard allows the user to login, sign up and navigate home. When a non-logged in user tries to join the society, the page will be directed to login scree.

## Login Screen

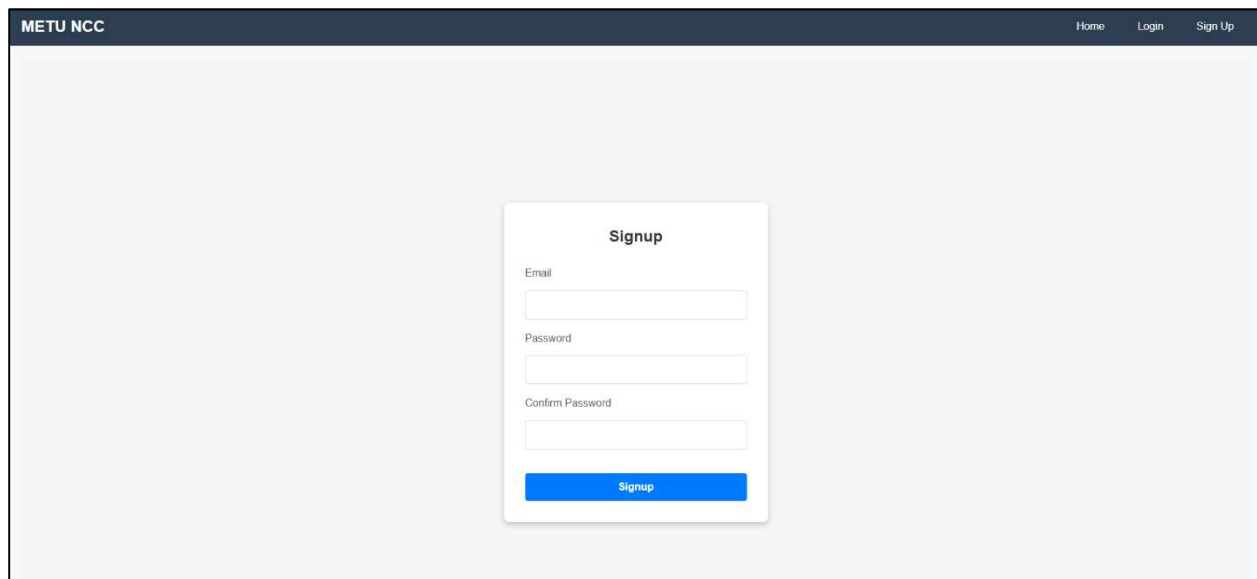


The image shows a web application interface for a login screen. At the top, there is a dark blue header bar with the text "METU NCC" on the left and three links: "Home", "Login", and "Sign Up" on the right. The main content area is light gray. In the center, there is a white rectangular box with a subtle shadow. Inside this box, the word "Login" is centered at the top. Below it, there are two input fields: the first is labeled "Email" and the second is labeled "Password". At the bottom of the box is a blue button with the text "Login" in white.

*Figure 3 Login Screen*

Login screen will allow the user to login to their account, and they will be directed to dashboard which will display different options according to the user role.

## Register Screen



The image shows a web application interface for a sign-up screen. It has the same header as the login screen, with "METU NCC" and links to "Home", "Login", and "Sign Up". The main content area is light gray. In the center, there is a white rectangular box with a subtle shadow. Inside this box, the word "Signup" is centered at the top. Below it, there are three input fields: the first is labeled "Email", the second is labeled "Password", and the third is labeled "Confirm Password". At the bottom of the box is a blue button with the text "Signup" in white.

*Figure 4 Sign up Screen*

If the user is unregistered they can register using the registered screen. Admin is responsible to assign roles for users.

# Admin Operations

## Admins Dashboard

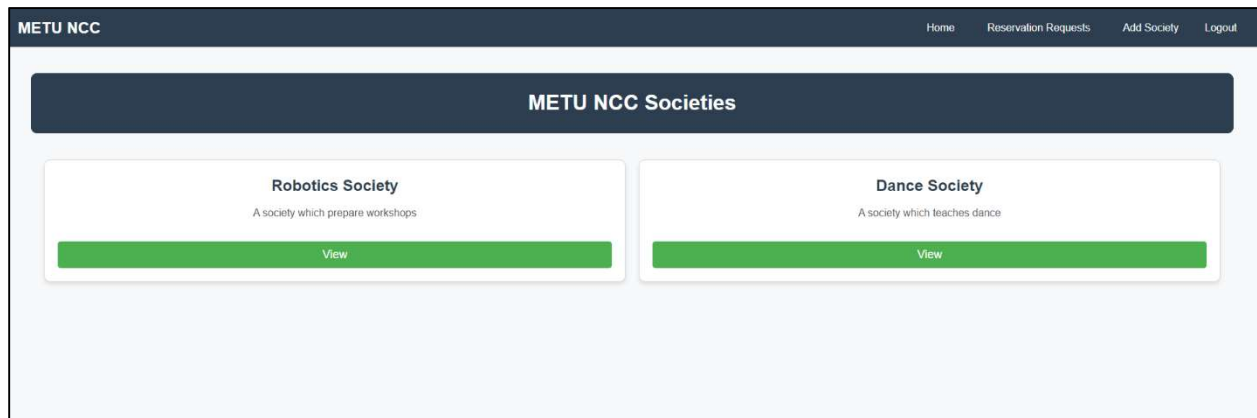


Figure 5 Admins Dashboard

Admins dashboard will show the available societies in the system. Moreover, it will show options for the admin to view reservation requests and add society.

## Admin View Society

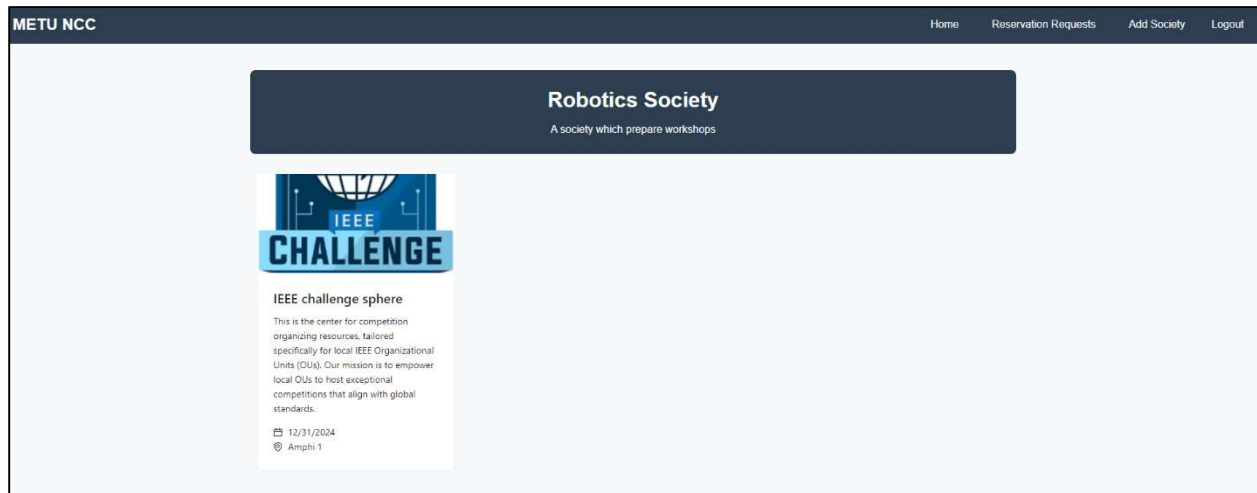


Figure 6 Admins View Society Screen

When the admin clicks on view society he will see the announcement related to the selected society.

## Reservation Request

The screenshot shows the 'Reservation Requests' page of the METU NCC system. The page has a dark blue header with the logo 'METU NCC' on the left and navigation links 'Home', 'Reservation Requests', 'Add Society', and 'Logout' on the right. The main content area is light blue and titled 'Reservation Requests'. It displays two reservation requests in a list. Each request is shown in a white card with a title, date, time, location, status, and president's email. Below each card are two buttons: 'Approve' (green) and 'Reject' (red).

Request Title	Date	Time	Location	Status	President Email	Actions
IEEE challenge sphere	2024-12-31	17:00:00	Amphi 1	pending	cagalay@metu.edu.tr	<button>Approve</button> <button>Reject</button>
gathering	2024-12-31	14:00:00	amphi 2	pending	eray@metu.edu.tr	<button>Approve</button> <button>Reject</button>

Figure 7 Reservation Request Screen

When the admin wants to check the reservation request, they need to click the Reservation Requests button on the top right of the screen. They will be directed to Reservation Requests screen which will show the list of requests with approve or reject buttons for each request.

## Add Society

The screenshot shows the 'Add a New Society' form in the METU NCC system. The page has a dark blue header with the logo 'METU NCC' on the left and navigation links 'Home', 'Reservation Requests', 'Add Society', and 'Logout' on the right. The main content area is light blue and titled 'Add a New Society'. It contains three input fields for 'Society Name', 'Description', and 'President Email'. Below the input fields is a large green button labeled 'Add Society'.

Field	Input
Society Name	<input type="text" value="Enter society name"/>
Description	<input type="text" value="Enter society description"/>
President Email	<input type="text" value="Enter president email"/>

Add Society

Figure 8 Add Society

This page will allow the admin to add society to the system. The admin should enter the details of the society, name, description and president email.

# President Operations

## President Dashboard



Figure 9 President Dashboard

The main difference for the president dashboard is Society and Request options at the header.

## Society

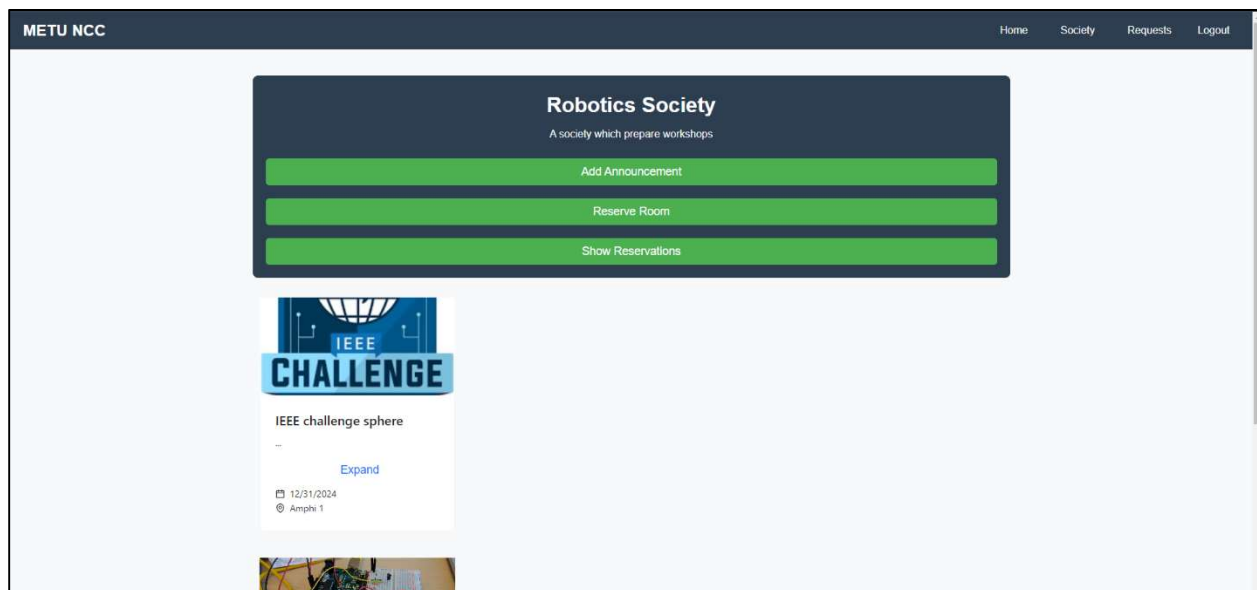
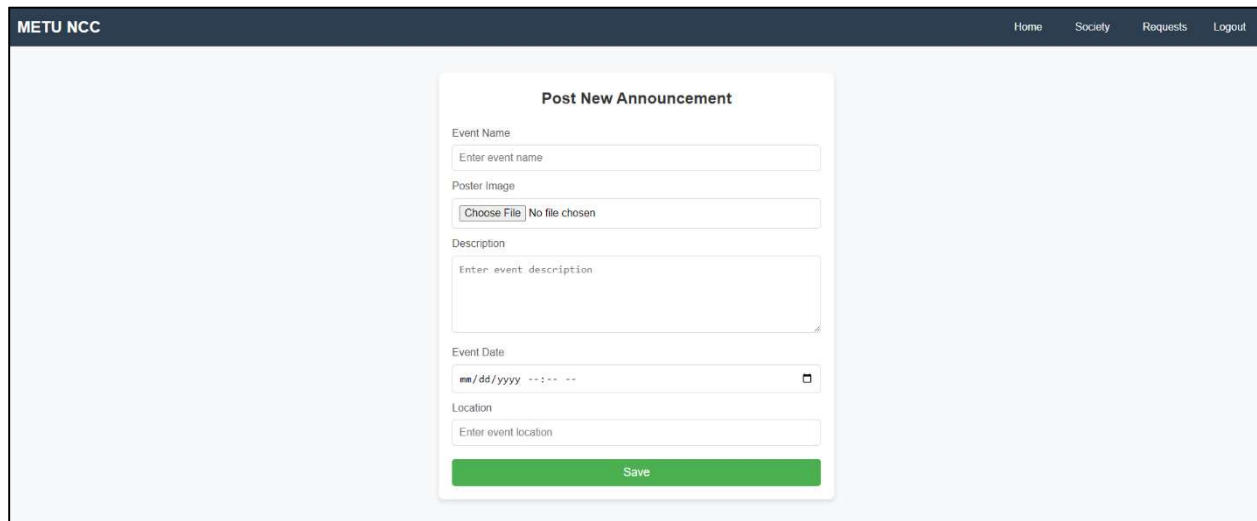


Figure 10 Society screen for president

The society button at the header will show the society the president is responsible for. The president will be able to add announcements, reserve rooms and show the previous reservations.



## Add Announcements

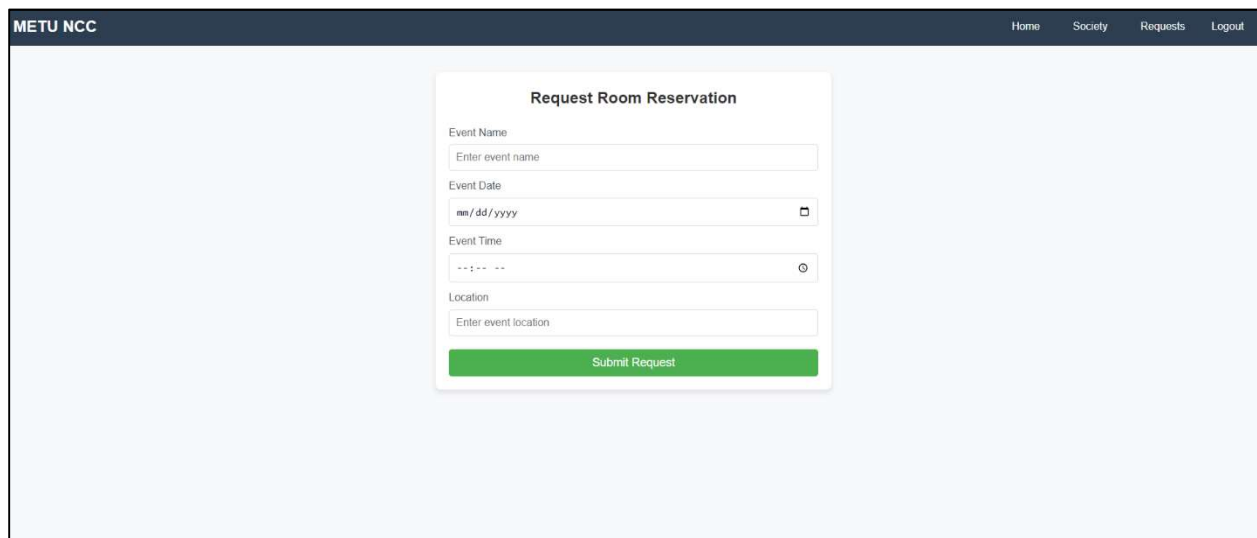


The screenshot shows a web application interface for METU NCC. At the top, there is a dark blue header bar with the text 'METU NCC' on the left and navigation links 'Home', 'Society', 'Requests', and 'Logout' on the right. The main content area is light blue and contains a white form titled 'Post New Announcement'. The form has the following fields: 'Event Name' (text input), 'Poster Image' (file upload button labeled 'Choose File' and 'No file chosen'), 'Description' (text area), 'Event Date' (date and time picker showing 'mm/dd/yyyy --:-- --'), and 'Location' (text input). A green 'Save' button is at the bottom of the form.

Figure 11 Add Announcements

To post an announcement, the president needs to fill out the announcement form.

## Room Reservation



The screenshot shows a web application interface for METU NCC. At the top, there is a dark blue header bar with the text 'METU NCC' on the left and navigation links 'Home', 'Society', 'Requests', and 'Logout' on the right. The main content area is light blue and contains a white form titled 'Request Room Reservation'. The form has the following fields: 'Event Name' (text input), 'Event Date' (date and time picker showing 'mm/dd/yyyy --:-- --'), 'Event Time' (time picker showing '--:-- --'), and 'Location' (text input). A green 'Submit Request' button is at the bottom of the form.

Figure 12 Room Reservation

To reserve a room, the president should fill out the room reservation form.

## Show Reservations

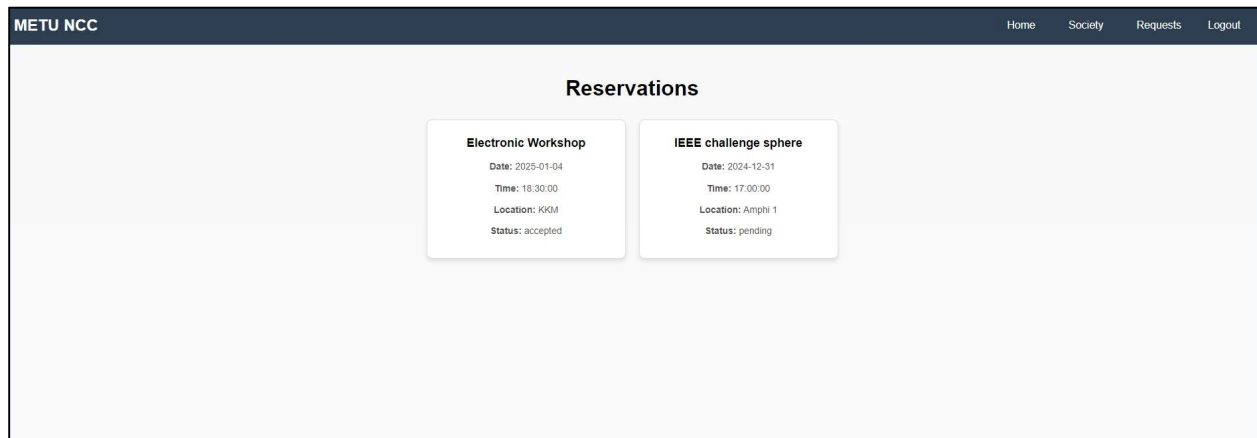


Figure 13 Show Reservations

Show reservation button will show the previous reservations done by the society president and their status (pending, approved, rejected)

## Requests

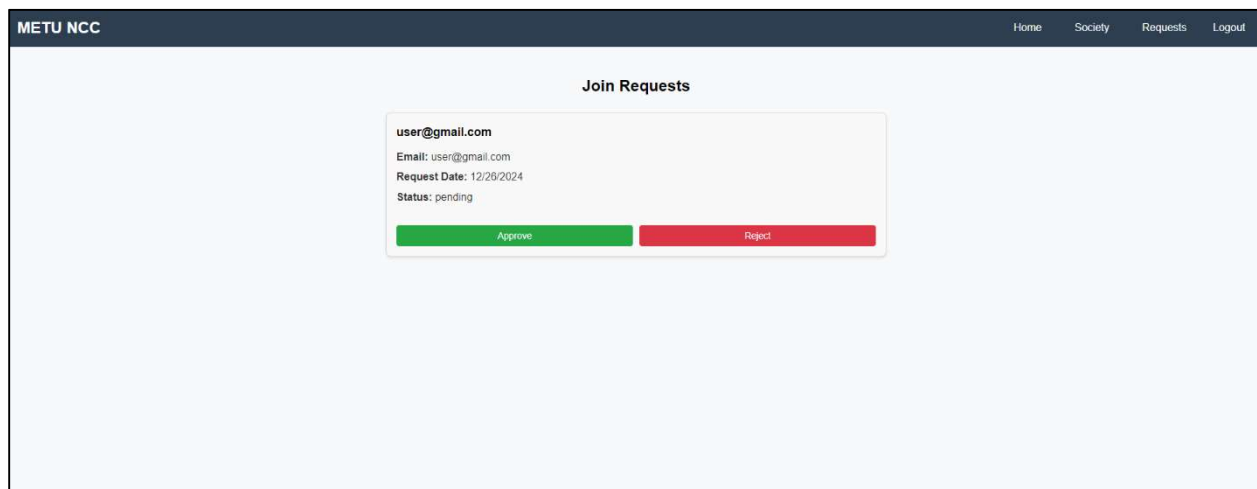


Figure 14 Join Requests Screen

Requests button in the header will show the users who requested to join the society with options to either approve or reject their request.

# User Operations

## User Dashboard

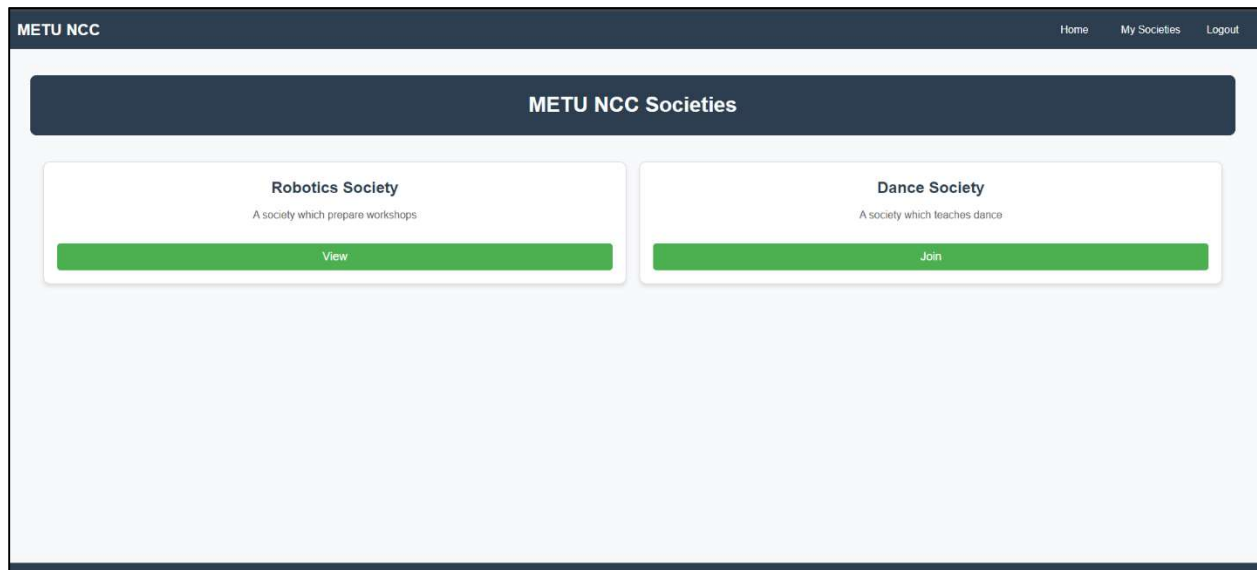


Figure 15 User Dashboard

User dashboard will show the available societies with an option to join the society. If the user is already a member in the society View option will be displayed instead of join. Moreover, in the header the user can see the societies they have joined.

## My Societies

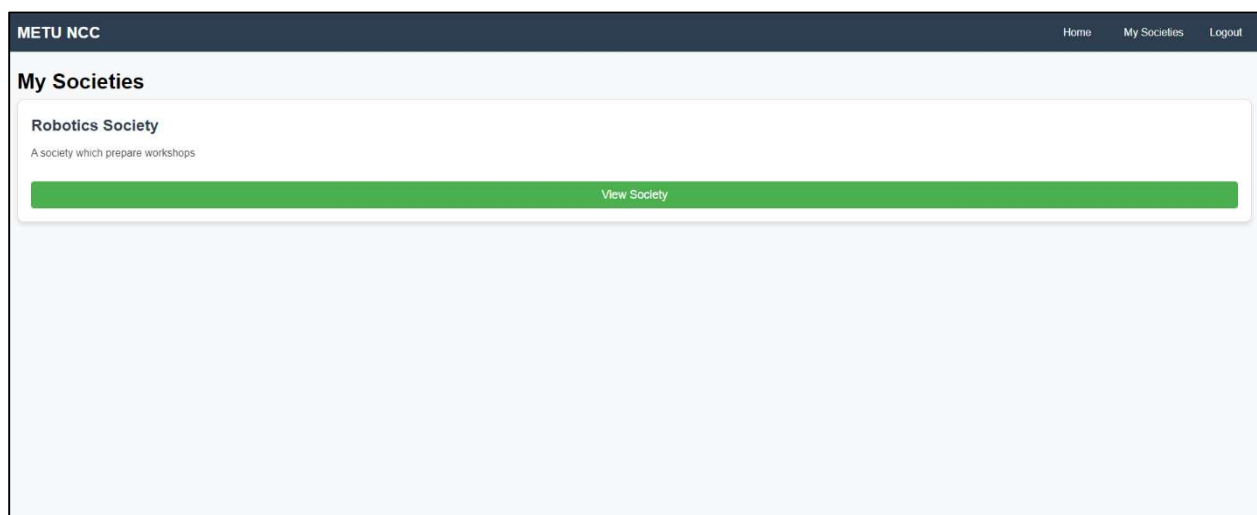
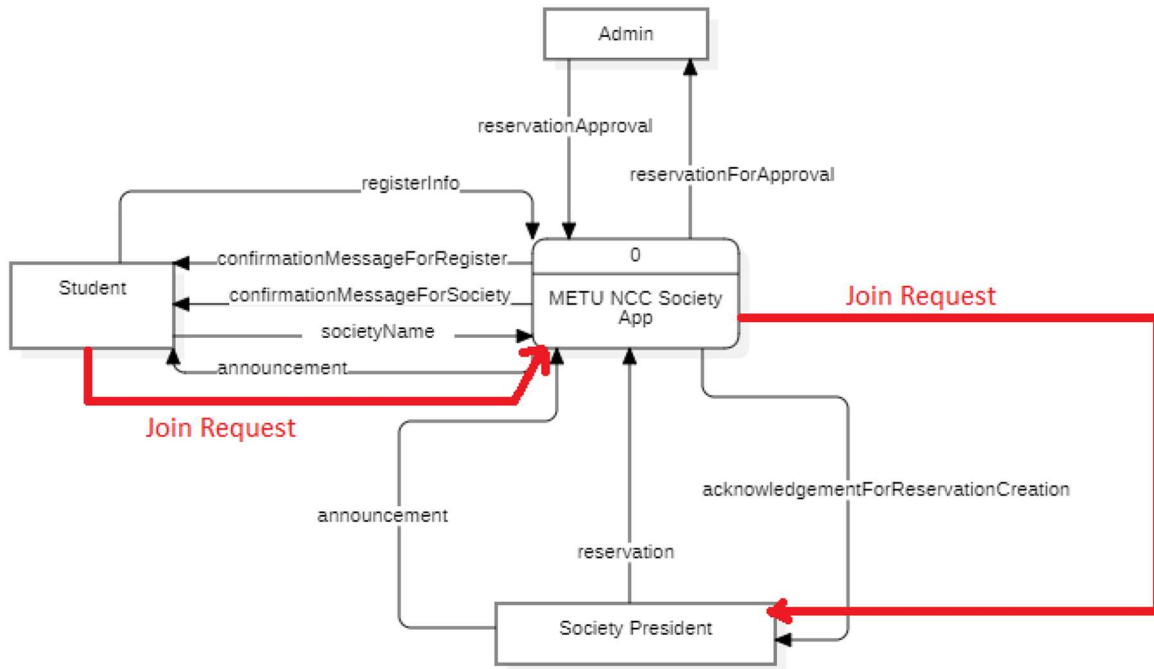


Figure 16 User Joined Societies Screen

This page will show the user the societies they have joined and their request was approved. They can also view the announcements by clicking on the view society button.

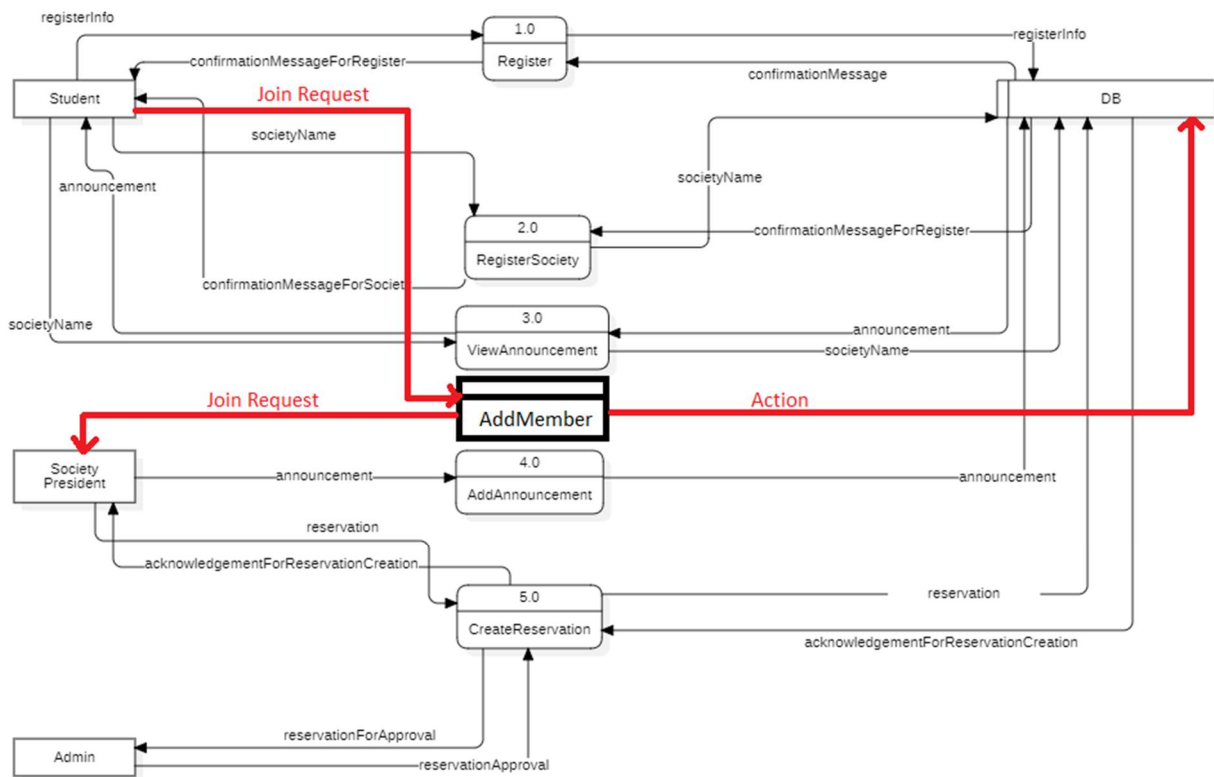
# Diagrams

## Context Level Diagram



After we started the project, we realized that adding join request could be much better to make our system more logical. Therefore, a student can send join request to society to be able to view announcements.

## Level-0 Diagram



This level-0 diagram shows the logic of the join requests. Firstly, a student can list all societies but cannot view the announcements of the society. To be able to view announcements, students should join the society (Register Society). Thus, students send a join request, and then this request is saved into the database. After that, the president of the society can list all join requests. The president can approve or reject the request, and this answer is accepted into the database. According to this answer, students can view their announcements or not.

Society president can add announcements create reservations. Admin can approve or reject reservation and it will be saved in database accordingly.

# List of Technologies

## 1- Front-end

Programming Language: JavaScript.

FrameWork: React

Extra: CSS, HTML

## 2- Backend

Programming Language: Java

FrameWork: JavaSpringBoot

DataBase: PostgreSql

## 3- Cloud Services:

AWS S3

AWS RDS

AWS EC2

AWS IAM

# Explanation and Difficulties

## Engin Eray Kabalak

I implemented backend APIs for Society, Announcement, Join Request, and Reservation Request. Additionally, I implemented the S3 service on the backend.

I also created IAM users and avoided using the root account to ensure better security. For the IAM user, I assigned only the necessary permissions. For example, I granted permissions to access and use S3 and generated an access key and a secret access key specifically for this IAM user.

If I were to rank the most challenging parts, the most difficult part on the backend was ensuring that when a president sent a reservation request, it was forwarded to all admins, and once approved, the president could see the approval status.

Adding the Amazon S3 service was also quite challenging due to dependency issues and implementing the S3 service in Java Spring.

Finally, handling the join requests that users sent for societies and ensuring that presidents could approve them was another challenging aspect. Other parts were relatively easier compared to these.

## **Ekrem Cagatay Goz**

I implemented back-end API for authentication and tokenization system. These are Login, Signup, JWT Authentication Filter. After these ones, I adjusted the security and application configurations to be able to run the authentication systems properly.

In addition to this, I deployed the website using EC2 server. Firstly, I build the back-end and front-end side. The front-end side build gave me a folder which includes index.html and other necessary files. I copied and pasted this folder into /var/www/html/ (NGNIX server location). After configuration, our website appeared in the IP in the browser. Our public IP is <http://18.192.66.96/>.

After deploying front-end, I built the Java Spring Boot application, and uploaded the jar file into my server. After installing dependencies, I run the jar file as a service in the local of machine.

The difficult part of this process was CORS error. Since my front-end runs on a public IP, my back-end was banning my front-end requests. After realized that, I changed the incoming request errors.

At the end of this process, our project is deployed.

## **Haya Arabi Katibi**

I primarily focused on implementing the graphical user interface (GUI) and the frontend of the application. My role included designing user-friendly pages for login, signup, and various society-related functionalities such as announcements, reservations, and join requests.

In addition to frontend development, I took on the responsibility of integrating AWS services for RDS to support the project infrastructure.

For RDS, I worked on configuring the database and linking it to the application. I set up an AWS RDS instance with appropriate security groups. After configuring the RDS instance, I updated the backend with the RDS endpoint and verified that all database-related functionalities, such as user authentication and society data management, worked properly.

One of the challenges I faced in integration was making sure the frontend and the backend are compatible while maintaining security. I encountered issues such as misconfigured security rules, which initially blocked database connections, and had to troubleshoot them to ensure proper connectivity which took a lot of time.

Despite these challenges, RDS integration was successfully implemented, enhancing the scalability and security of the project.

# Project Statistics

## Time Frame

### October 17 - November 12

We Completed Proposal 1 in this period. All of us worked on the proposal

No implementation in this time frame.

### Nov 22- December 4

We Completed Proposal2 in this period. All of us worked on the proposal.

**Engin Eray Kabalak:** Added controllers, services, repositories, and entities for Societies and Announcements. - Implemented the following APIs: - Get all societies - Get society by ID - Get all announcements - Get announcement by ID

**Ekrem Cagatay Goz:** Added authentication logic and controller, services, repositories, entities for User. Created these endpoints: /login, /signup

**Haya Arabi Katibi:** added the missing pages such as: Admin Page, Announcement Form page, Edit Event page, Join Society Request page, My Societies page and finally Add Society page.

### December13- December 26

We Completed Final Report. All of us worked on the report. The website is completed, it's running on server now.

### Engin Eray Kabalak :

I worked on several backend features to improve the functionality and user experience of our application. For the Announcements API, I added the ability to fetch announcements by society ID and integrated Amazon S3 for handling file uploads, allowing users to include images in their announcements. I also updated the AnnouncementForm to make creating announcements with images seamless and introduced **Api.js** to centralize API calls, making the code more organized and easier to maintain. Additionally, I enhanced **Dashboard.js** and **SocietyPage.js** to fetch societies and related announcements directly from the database for a smoother user experience.

On top of that, I implemented APIs for managing join requests and reservation requests. The join request APIs let users request to join societies, with options for society presidents to approve or reject them. For reservations, I created APIs that allow society presidents to make reservation requests and admins to review and process them. These features simplify managing memberships and reservations, making the application more functional and user-friendly.



### **Ekrem Cagatay Goz:**

- 1- **Integration of back-end and front-end:** I connected two sides to each other. To be able to do that, I created REST requests and then took responses to these requests. According to the frontend developed by Haya, I embedded these answers into the front-end side. Also, there were some needs to be able to send correct requests. Therefore, I added back-end functions as well.
- 2- **Deployment of website:** After finishing our project, I was responsible for deploying front-end and back-end sides. Using AWS EC2 service, I created a server computer and then I built the two sides. At the end, I uploaded and run these builds in service computer, and right now, it runs in public IP.

### **Haya Arabi Katibi:**

In the final phase of the project, I set up and integrated AWS RDS as our main database. I configured the instance with the right settings, including the database engine and security groups, and made sure it could only be accessed securely by our backend. After connecting the backend to the RDS endpoint, I tested key operations like user authentication, managing societies, and handling reservations to ensure everything worked smoothly. Finally, I optimized the setup to improve performance and reduce delays, ensuring the database was reliable and efficient for our application.

I thoroughly tested the system to ensure data consistency and reliability across key operations like creating announcements, processing join requests, and managing reservations. Additionally, I optimized the connection settings to reduce latency and improve overall performance, ensuring smooth and efficient database interactions.

## **Number of Lines Code**

### **Back-end**

Programming Language:

Java, Spring Boot Framework: Approximately 1500 Lines of code Java

### **Front-end**

Programming Language:

CSS: 512 LOC

Html: 20 LOC

JavaScript: 1396 LOC

Approximately 1900 lines of code

# Memory Requirements

## **Front-End (React.js)**

**Development:** ~512 MB–1 GB for the local dev server and build process.

**Production:** Minimal overhead under a static server (e.g., Nginx), typically ~512 MB total.

## **Back-End (Spring Boot)**

**Development:** ~512 MB–1 GB (plus memory for your IDE).

**Production:** Start with 1–2 GB for light traffic; scale up (2–4 GB or more) for higher loads.

## **Database (PostgreSQL on AWS RDS)**

**Small Deployment:** 1–2 GB RAM (e.g., db.t2.micro) for low traffic.

## **AWS S3**

**Uploads:** Minimal extra overhead (50–100 MB) if files are buffered in memory before streaming. S3 itself is a managed service with no direct RAM usage on your side.

## **AWS IAM**

**No additional memory usage;** AWS manages this service entirely.

# Database

PostgreSQL Created using RDS.

Maximum Size is 100GiB.

Instance	
Configuration	Instance class
DB instance ID society-app	Instance class db.t4g.micro
Engine version 16.3	vCPU 2
RDS Extended Support Disabled	RAM 1 GB
DB name society	Availability

Figure 17 Database Configuration

Storage
Encryption Enabled
AWS KMS key <a href="#">aws/rds</a>
Storage type General Purpose SSD (gp2)
Storage 20 GiB
Provisioned IOPS -
Storage throughput -
Storage autoscaling Enabled
Maximum storage threshold 1000 GiB
Storage file system configuration Current

Figure 18 Database Storage

# References

1. AWS RDS: Amazon Web Services. (n.d.). *Amazon RDS Documentation*. Retrieved from <https://docs.aws.amazon.com/rds/>
2. AWS S3: Amazon Web Services. (n.d.). *Amazon S3 Documentation*. Retrieved from <https://docs.aws.amazon.com/s3/>
3. AWS EC2: Amazon Web Services. (n.d.). *Amazon EC2 Documentation*. Retrieved from <https://docs.aws.amazon.com/ec2/>
4. AWS IAM: Amazon Web Services. (n.d.). *AWS Identity and Access Management (IAM) Documentation*. Retrieved from <https://docs.aws.amazon.com/iam/>
5. Spring Boot: Pivotal Software, Inc. (n.d.). *Spring Boot Documentation*. Retrieved from <https://spring.io/projects/spring-boot>
6. React: React Team. (n.d.). *React Documentation*. Retrieved from <https://react.dev/>