MIDDLE EAST TECHNICAL UNIVERSITY
NORTHERN CYPRUS CAMPUS

Computer Engineering Program

CNG 495

CLOUD COMPUTING

FALL 2025- TERM PROJECT FINAL REPORT

WashMate APP

Team Members:

Nazlıcan Taviş- 2751741

Nisa Sağdıç- 2751691

Fatih Demirbilek – 2526234

WashMate App

The aim of this project is to develop a cloud-based Laundry Management System that enables students to efficiently manage their laundry schedules through a user-friendly web site. The students will be able to register in our system with their student number, email, name, surname etc. Their email adresses will be used to send notifications. Then they will be able to log in to view the dashboard of users. In the user dashboard, they can view the current status of all available washing and drying machines, whether they are available, in use, reserved or disabled in a table format. Users can reserve available time slots and machines so they will not waste their time waiting for a machine or creating conflicts to reserve. After they went to wash their clothes, they will state that in the interface. If a user books a time slot but fails to attend three times, the system will issue a warning. Additionally, if a user books a slot but does not show up, the slot will be reopened after five minutes. Every student will be able to see machine's status(Reserved/Avaliable/Out of order/In Use). When they booked available machine, when the machine starts and finishes it washing they will receive a notification via their email.

The application is designed to make the laundry process more organized and accessible within students and dry cleaner service providers by offering real-time machine availability updates and online booking features, it significantly improves user convenience and resource utilization. In addition, users can change the machine status as disabled directly through the system, so admin can see those problems and solve them immediately.

Administrators will have access to an admin dashboard where they can add or remove machines. Also, the admin will have right to reject or approve the student registrations. They will also be able to see machine statistics, such as which machine has how many drying/washing cycles, etc.

The Smart Laundry System integrates cloud technologies to deliver a reliable, and efficient solution for students who use common machines. By hosting the data and services in the cloud, large amounts of user and machine data can be managed easily and the system ensures high availability.

Benefits and Novelties of the Project

WashMate is a effective cloud-based solution designed to help people manage jointly used laundry facilities in a university setting. Its biggest advantage is the time savings it provides, allowing users to have real-time information about machine availability and reserve machines that are available in advance. This reduces queues and conflicts that can arise if people violate the machine start-up time rule specified in the rules. Users can also report machines that are out of service to the administrator, so that malfunctioning machines can be repaired quickly and other users can instantly monitor machines that are out of service. This allows machines to have their malfunctions repaired faster and more reliably than organizations that do not use this application.

From an administrative point of view, the role of WashMate is in enabling the administration of machines and users from a central point, where the admin can approve registrations, track the status of the machines, and review statistics for better maintenance. Furthermore, laundry service providers using this system will be able to monitor their machines in real-time more quickly and securely than other providers, thus saving time on maintenance, replacements, and other operations.

What makes the project novel lies in the serverless and cloud-native approach. This comes about because the application utilizes services such as Authentication, Firestore and Hosting provided by Firebase. Such an approach provides real-time data synchronization, high availability, and scalability without server management. Additionally, the automated notification system by using Email.js makes the application even more user-friendly.

Similar Existing Projects

- **Laundry booking system** ,GitHub: https://github.com/lundsnation/laundry_booking
- **Laundry reservation app,** GitHub: https://github.com/TKYK93/Laundry-booking
- **Laundry booking,** GitHub: https://github.com/emilieanthony/laundry-booking

Project Components and Functions

1. Client-Side Web Application (Frontend)

- The front-end is developed using HTML, CSS, and JavaScript. It provides the graphical user interface for both users and administrators.
- Users can sign up, login, view available/reserved/out of order/in use machines, reserve avaliable machines, report issue on machines, and see rules and contact informations.
- The system also provides administrators with a dedicated dashboard where they can approve users, add or remove machines, see machine usage statistics, view available/reserved/out of/in-use order machines, and change the status of the repaired machines as "available" . Administrators also have access to the Admin System Controls panel. From this panel, they can create test machines, take backups, reset the database, and no shown warnings to users who do not attend 3 times to reservations.
- The frontend communicates securely with Firebase services using the Firebase JavaScript SDK.

2. Authentication Component

- User authentication is handled by Firebase Authentication.
- It supports secure email- and password-based login and registration. Once the admin approves the new user, a confirmation email is sent to the user, and the user can log in to the system. Since this system was designed for METU NCC, students will log in using their METU passwords, therefore there is no password reset method.
- User roles (student or admin) are identified using role information stored in the system, ensuring proper access control to pages and operations.

3. Database Component: Firebase Firestore

- Cloud Firestore is used as the main NoSQL database.
- It stores user information, machine data, booking records.
- Firestore provides real-time data synchronization, so any change—such as machine status updates —is immediately reflected on all connected clients.
- It provides highly available and scalable data storage solution without requiring manual server management.

4. Booking System

- The booking system allows students to reserve washing or drying machines.
- Reserved/Avaliable/Out of/In-use order machines are visible to all users, preventing conflicts.
- If a user does not attend a booked slot, the reservation is automatically reopened after 5 min.
- Repeated unattended reservations (3 times) trigger warning mechanisms to ensure fair usage of the system.
- The user uses a button to notify the administrator if the machine is malfunctioning. This allows the administrator to see the faulty machine, and when the user presses the button, the machine goes into an "out of order" state.
- Also, user can start washing/drying immadiately if machine status is avaliable.

5. Notification Component

**Firebase charges a fee for implementing this service so, we used Email.js for emails.

- To ensure the project continues to run on the free tier, we integrated the EmailJS service, which enables sending emails without setting up a backend mail server.
- EmailJS is called via triggers on the frontend and sends "Started/Finished" emails to the user using pre-made templates.
- EmailJS delivers the email by forwarding the request to the relevant provider through its own infrastructure.
- A detailed explanation and tutorial for this section have been included in the report.
- EmailJS is a cloud service/third-party SaaS.

6. Hosting and Deployment Component

- The application is hosted using Firebase Hosting.
- Frontend files are served through a global Content Delivery Network (CDN).
- Firebase Hosting provides HTTPS by default, ensuring fast loading times and high availability, and integrates seamlessly with Firebase Authentication and Firestore.

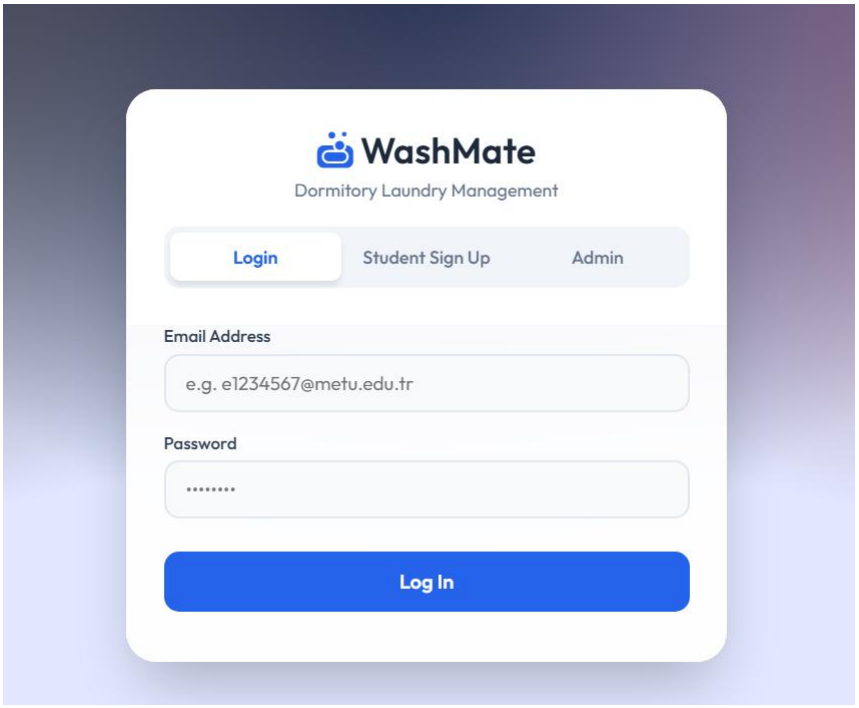
Overall, these components together form a fully cloud-native, serverless system that enables real-time interaction, efficient resource management, and a user-friendly experience for both students and administrators.

Utilized Cloud Services

Cloud Service	Provider	Purpose in the Project
Firebase Firestore	Google Firebase	NoSQL database for storing users, machines, and appointments with real-time synchronization
Firebase Hosting	Google Firebase	Deployment of the frontend web application with CDN support and HTTPS
External Cloud Email Service	Email.js SaaS (External service)	Sending email notifications for completed cycles and registration approval
Firebase Authentication	Google Firebase	Secure user registration and login, email/password authentication, role-based access control (student/admin)

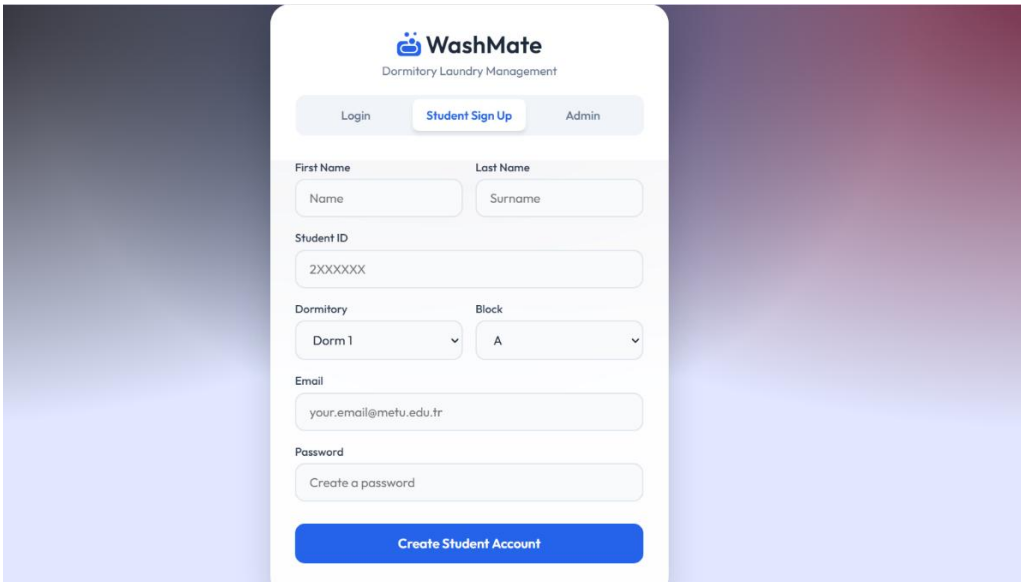
User Manuel:

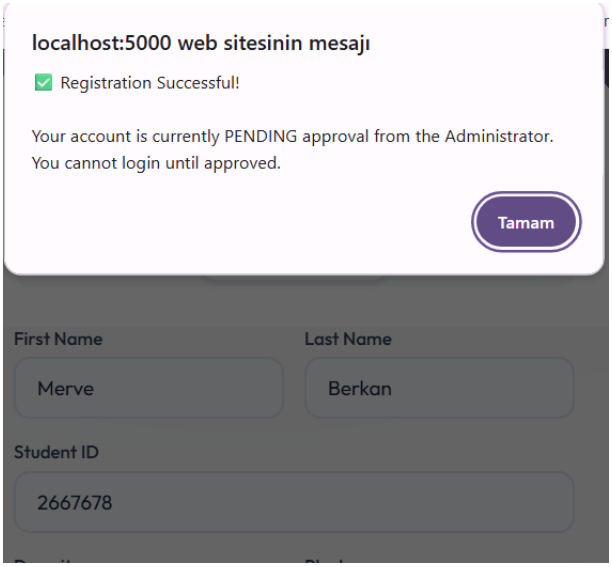
- Administrators and users use a shared login page to access the system.

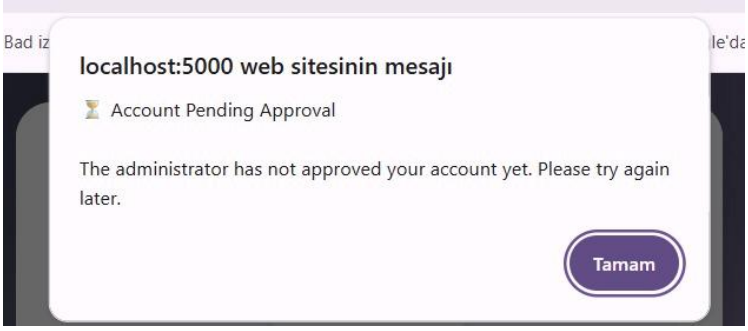


Students Side:

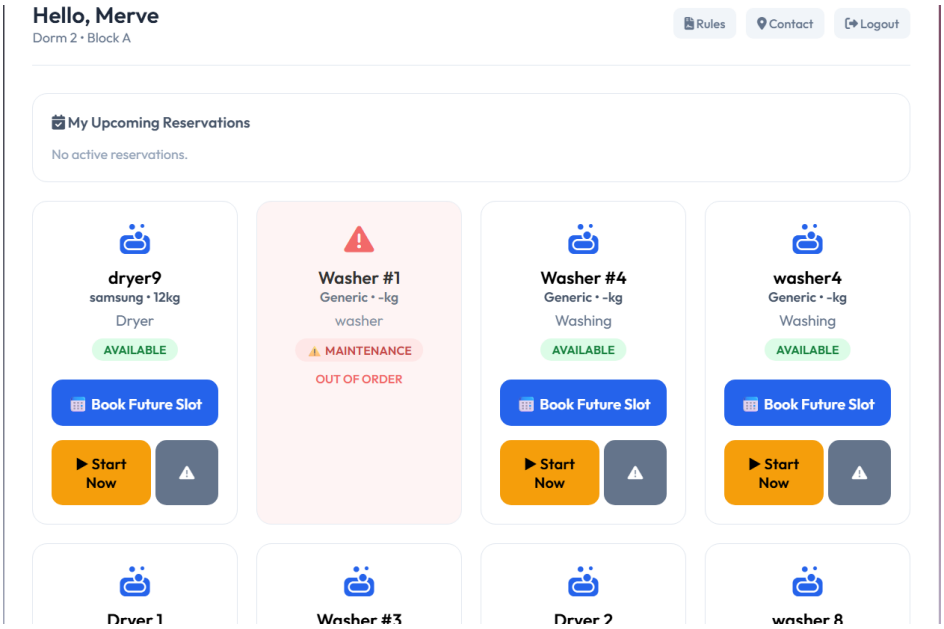
- First, you need to create an account to use laundry system. The user enters the information requested by the system and creates an account. However, the user cannot log in to the system until the administrator approves the new user.



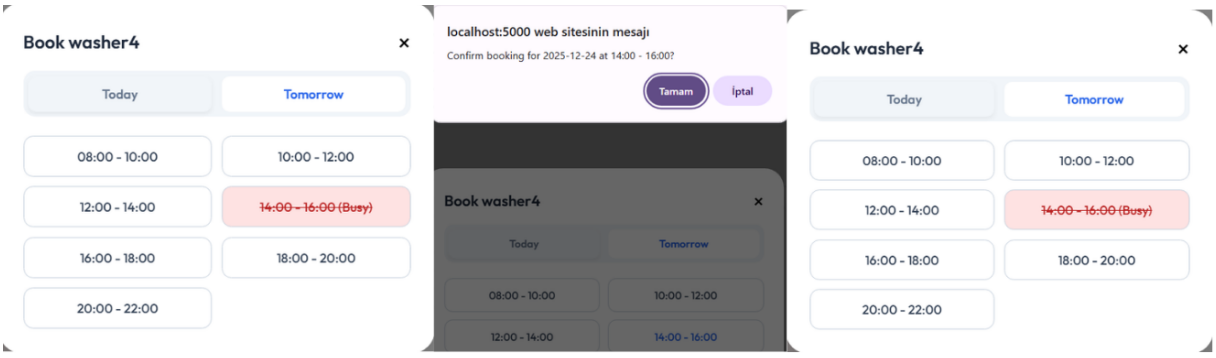


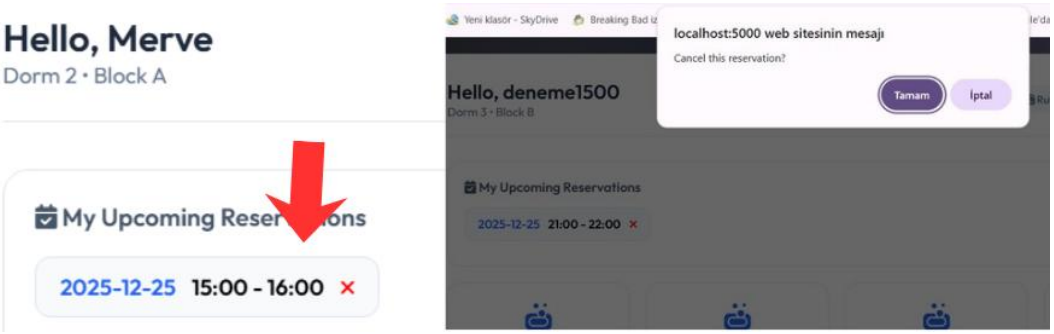


- After a user approved by the administrator logs into the system, they can view the user dashboard. On this screen, the user can see all machines and their statuses. From this screen, the user can easily perform operations such as timeslot booking and machine problem reporting. The user can also read laundry rules and access company contact information.

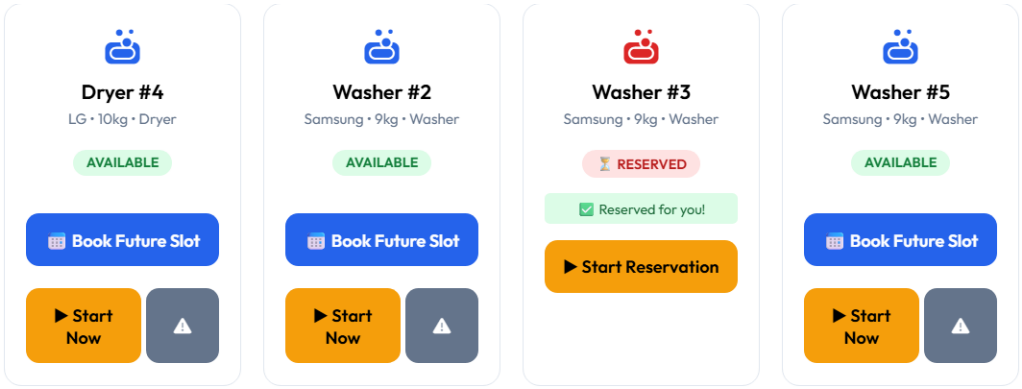


- Users can book specific time slots for today and tomorrow and view these reservations from their main screen. Also, users can cancel their reservations.

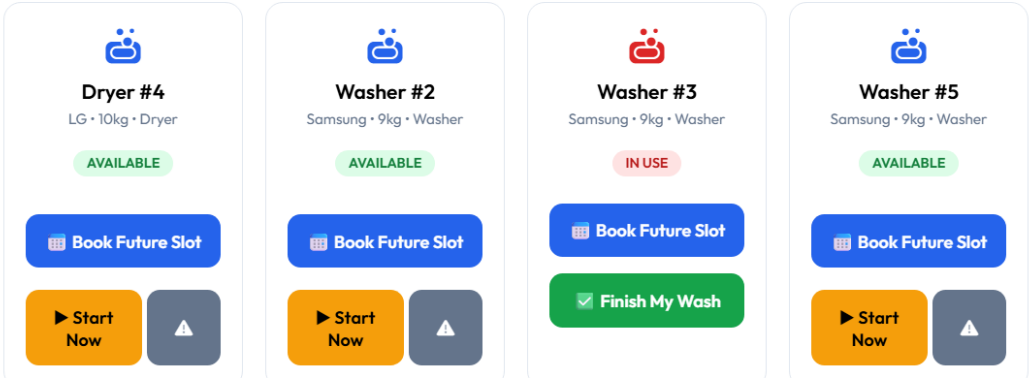




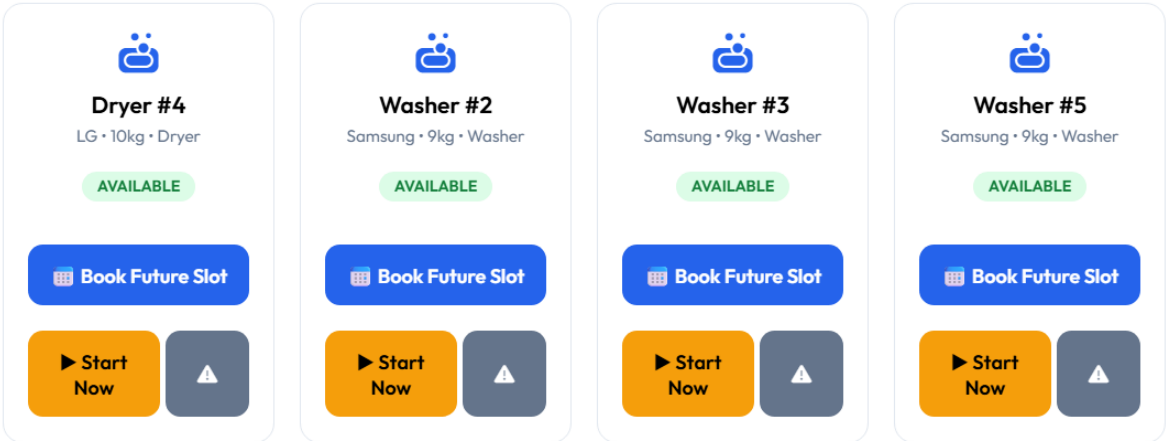
- **No Show Rule:** When users' appointment time arrives, the status of the machines they booked automatically changes to reserved. Users must prove their attendance by pressing the "start reservation" button within five minutes. In this case, the machine status changes in-use for all users. If they do not press the "start reservation" button within five minutes, the system recognizes that the user has not attended the appointment, and the machine becomes available again.



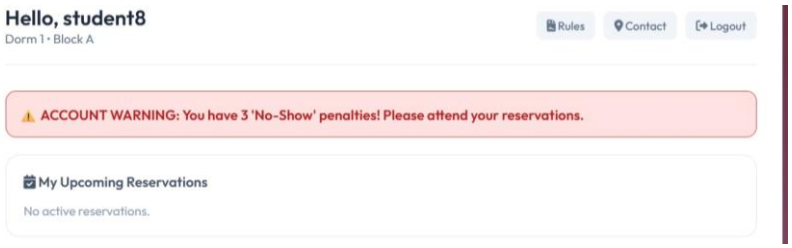
- **Pressed “Star Reservation” Button**



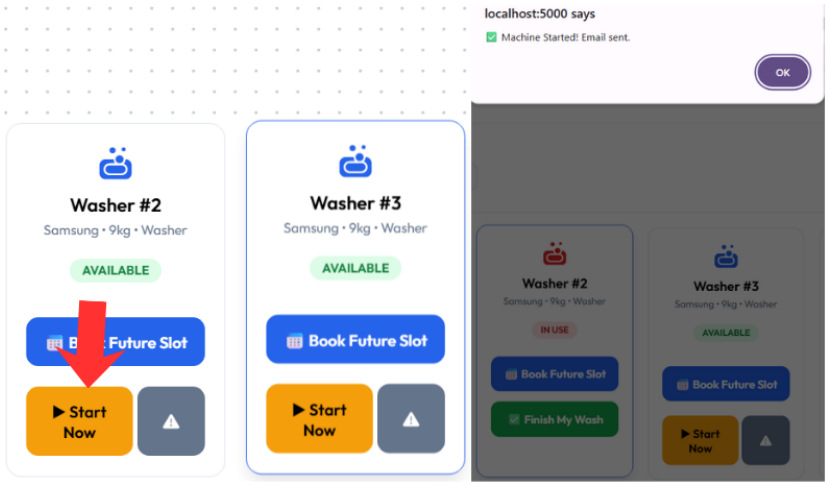
- **Unpressed “Star Reservation” Button:** After 5 min



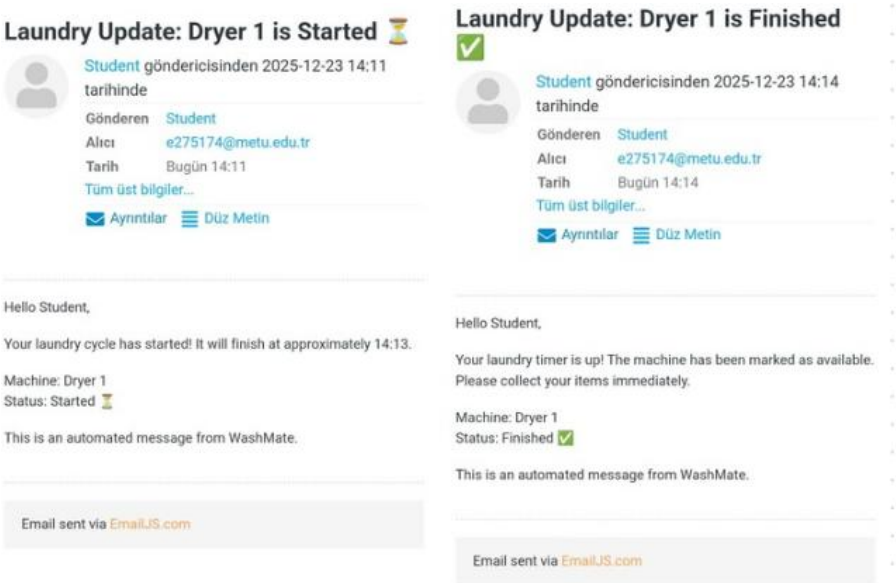
- If the user misses their appointment three times, meaning they don't press the "start now" button three times, the system will send a warning.



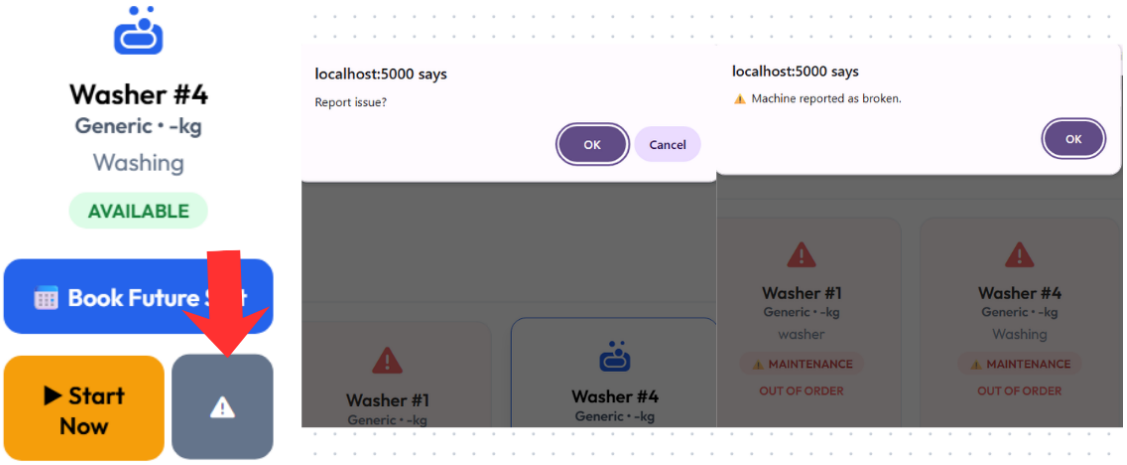
- If a user wishes to immediately use an available machine that is not currently in use, they can indicate that they are using that machine by pressing the "start now" button. This allows them to use available machines in place of users who missed their appointments. This is another feature that ensures fairness in the appointment system.



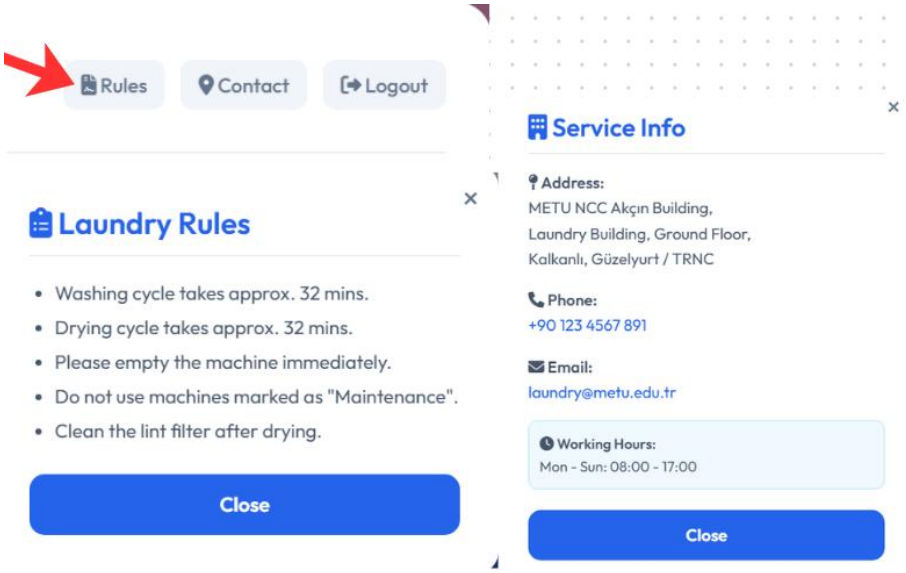
- The user receives a notification email when the machine starts and stops.



- Users use the report button to report if the machines are not working or are malfunctioning. Such problematic machines immediately appear as out of order for other users.

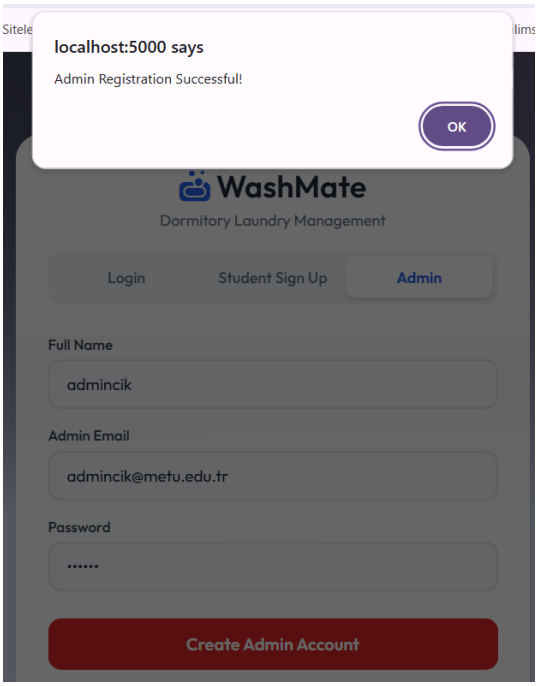


- Users can view laundry rules and service information from the homepage.

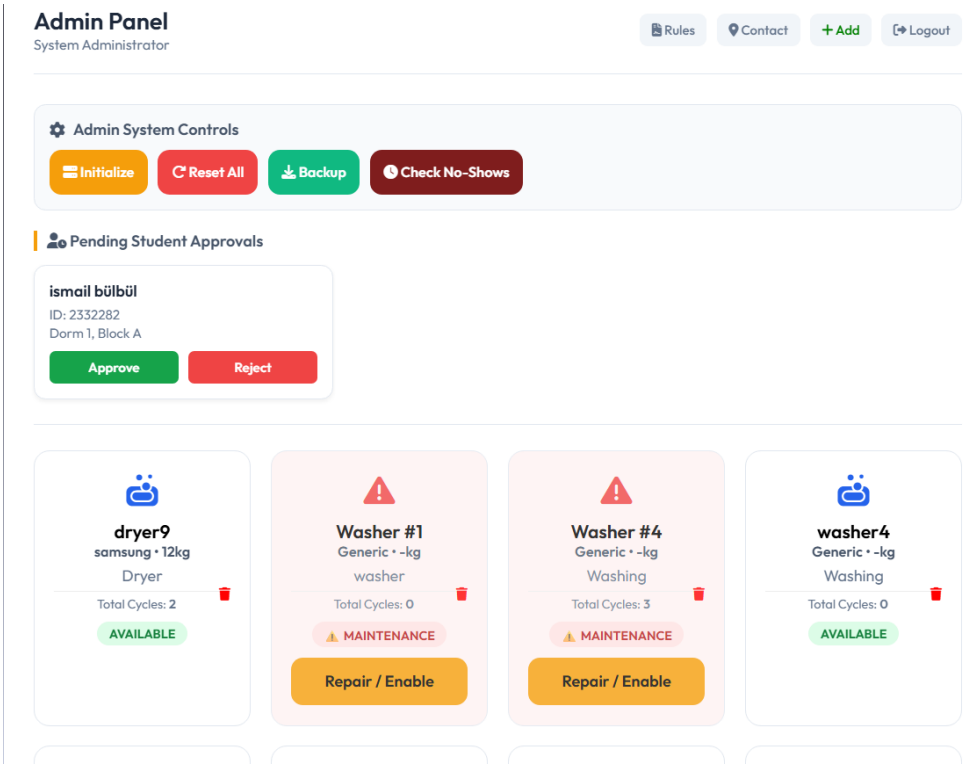


Admin Side:

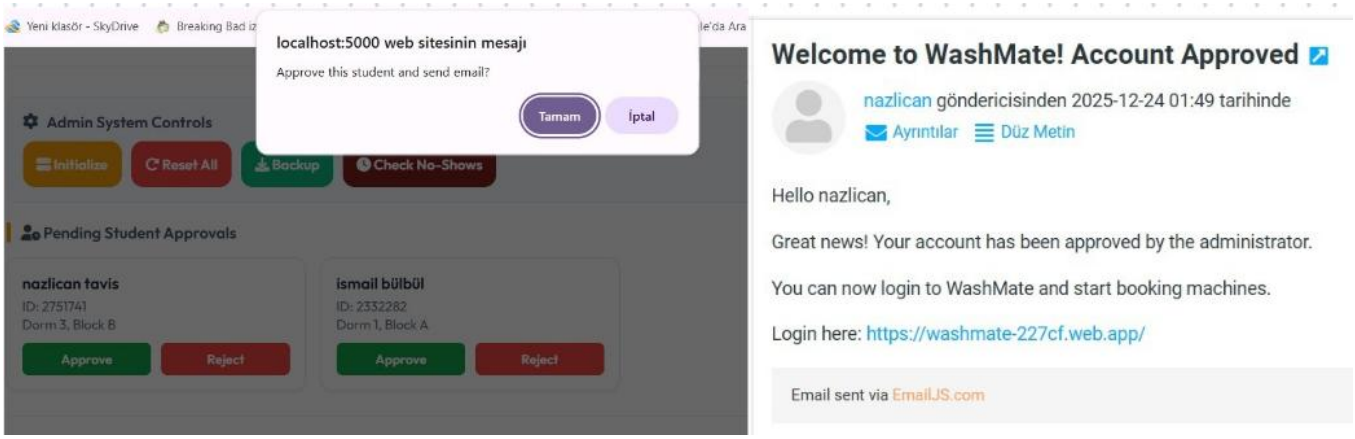
- First, admins need to create an account to use laundry system. The admin enters the information requested by the system and creates an account. The system has seperate admin sing-up page.



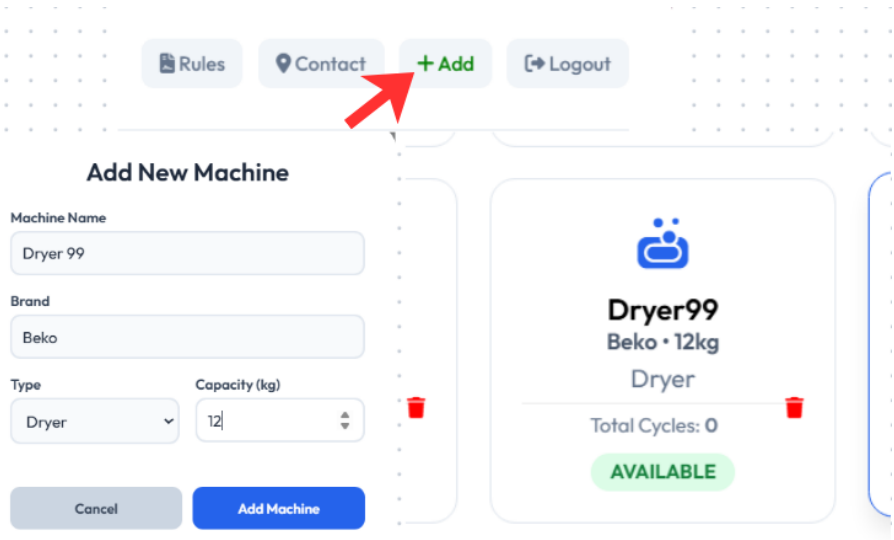
- After logging into the system, an administrator can view the admin dashboard. On this screen, the administrator can see all machines, their status, and total cycles. The administrator can add and delete machines, and make malfunctioning machines available again. The administrator can approve or reject newly registered users. Finally, the administrator can create test machines, back up the system, and reset the status of all machines using the system control panel.

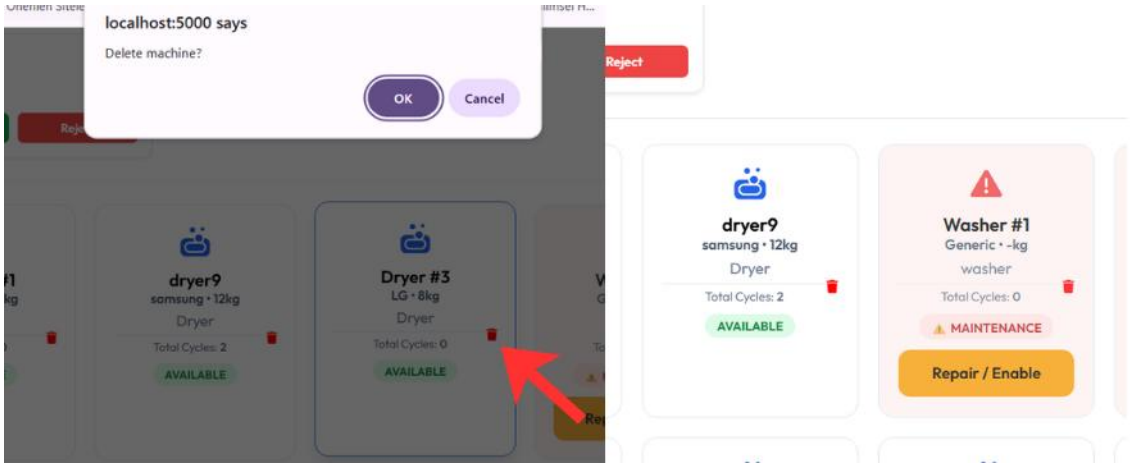


- Admins can view pending user approvals for new users on the homepage and approve or reject them. If the admin approves the user, an email will be sent to the user.

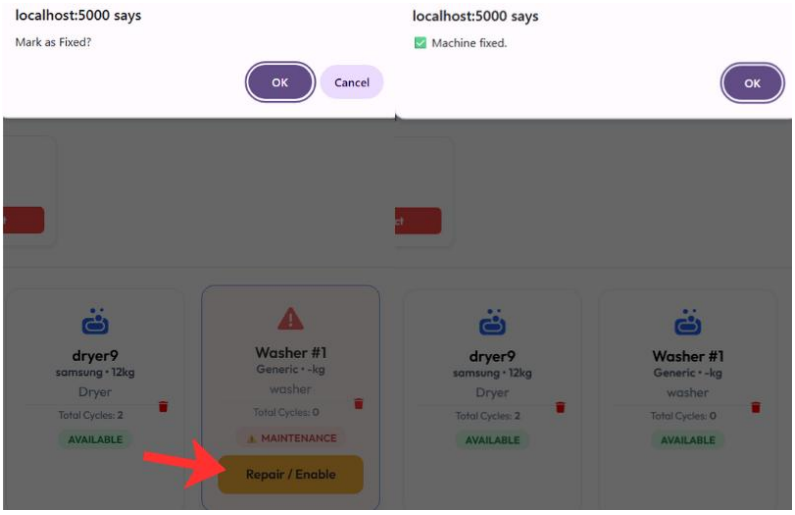


- The administrator can add machines by entering the necessary machine information into the system and delete existing machines.





- Admins can make out-of-order machines available for users to use after they have been repaired.

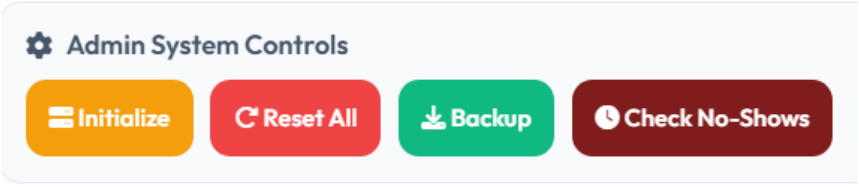


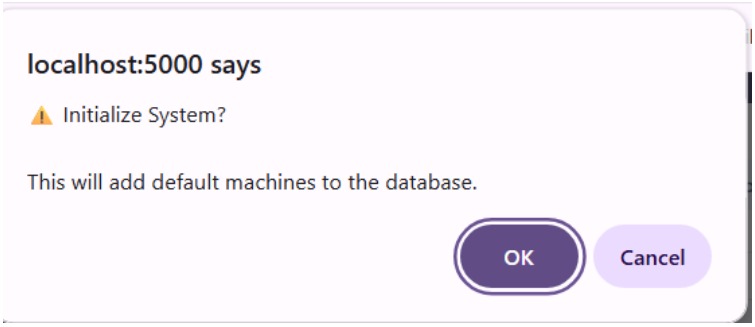
- The administrator can perform initialization, backup, and reset operations via the system control panel;

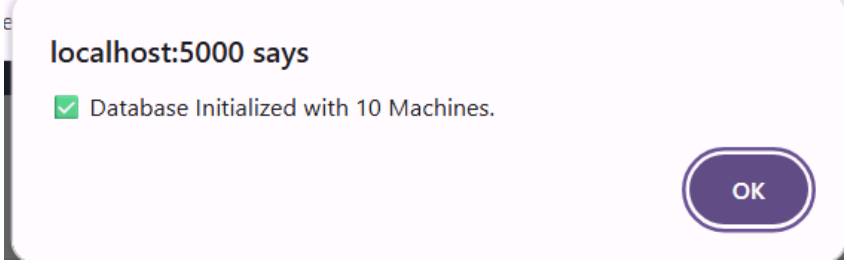
  Initialize: Create 10 test machines

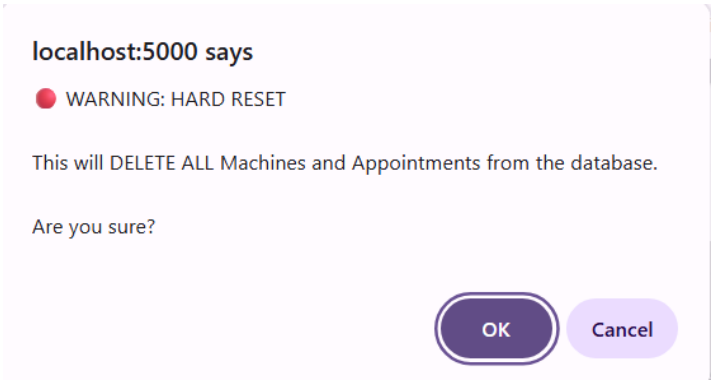
  Backup: Download a backup of all data to local computer as json file.

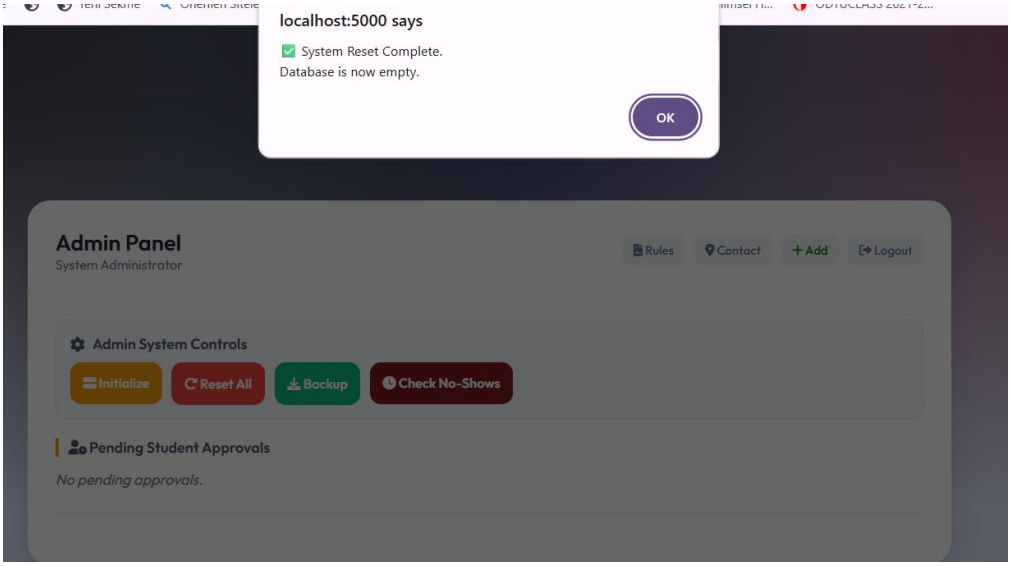
  Reset: Delete all machine and appointment data.

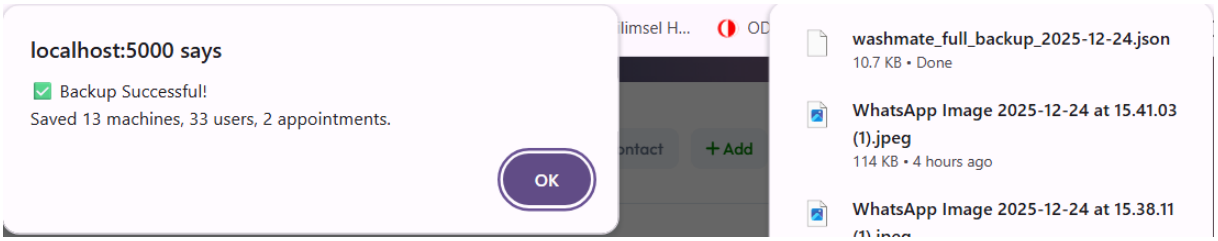












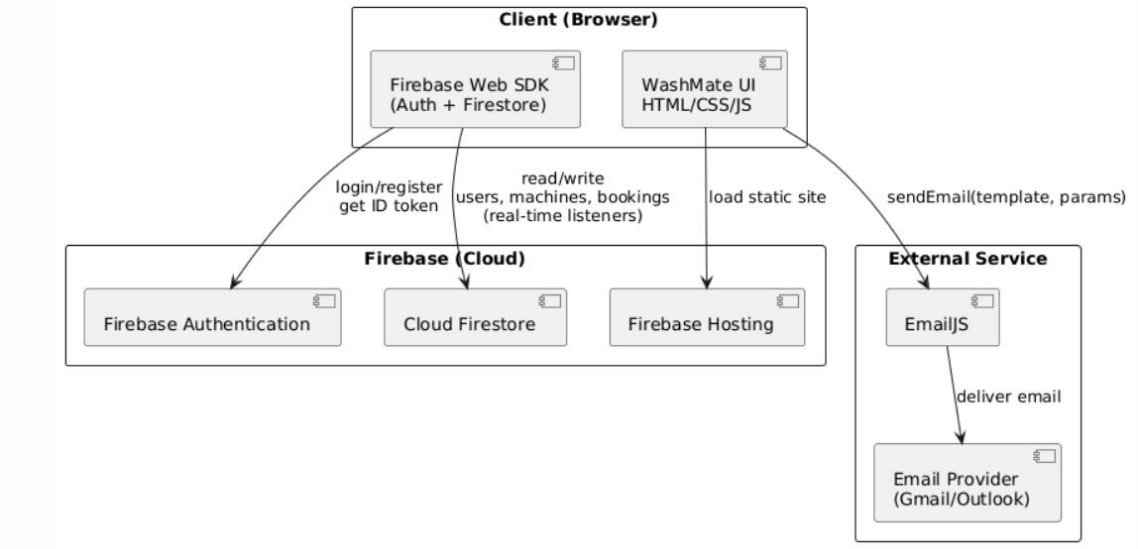

```
  {
    "id": "HGWIL05O96iu4HBSVNoS",
    "type": "Washer",
    "name": "Washer #4",
    "status": "available",
    "capacity": "9",
    "brand": "Samsung",
    "usageCount": 0
  },
  {
    "id": "LPgnGEhcruvoXcF4YffL",
    "usageCount": 0,
    "name": "Washer #5",
    "status": "available",
    "capacity": "9",
    "type": "Washer",
    "brand": "Samsung"
  },
  {
```


St 1, Süt 1 | 10.960 karakter | Düz metin

System Architecture Diagram

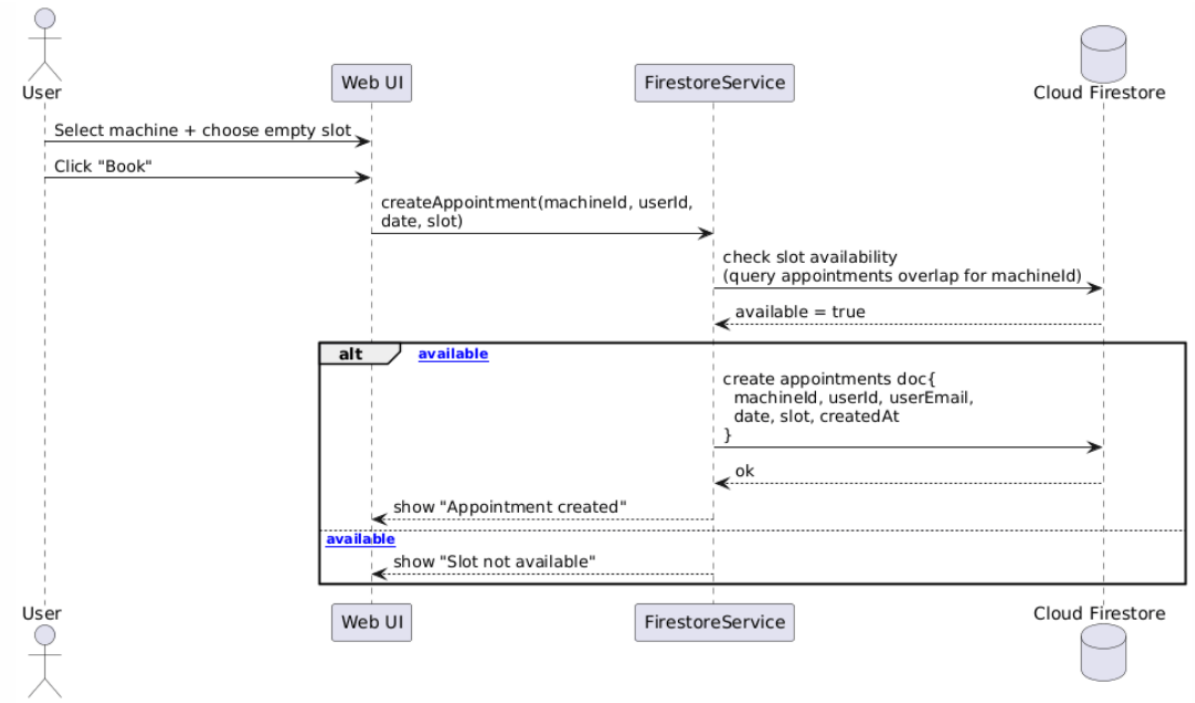
This diagram generally illustrates the system's connection to cloud services and email services:

- The application is statically served on Firebase Hosting; user authentication is done with Firebase Authentication.
- Machine status, user information, and appointments are stored in Cloud Firestore, and the UI is updated in real-time with listeners.
- When the user starts and stops the machine, the client-side notification layer sends an email via EmailJS based on the machine status.

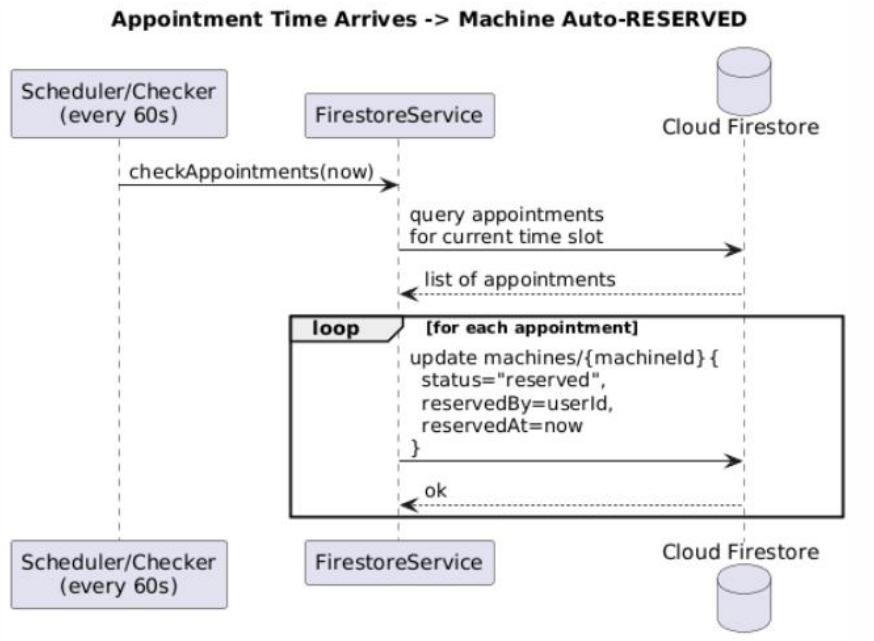


Laundry Washing with Appointment Case Sequence Diagram:

Slot Booking: Purpose: To allow the user to select an available time slot and create an appointment. The user selects a machine and chooses an empty slot in the UI, then presses the Book button. The UI calls the appointment creation code in FirestoreService. The service checks for availability to see if there are other appointments for the same machine at the same time. If the slot is available, a new appointments document is added to Firestore (machineId, userId, date, slot, createdAt...). The UI confirms to the user with "Appointment created". We show that the slot is occupied by drawing a line over it in the UI, but in the diagram, the case is expressed as "Slot not available" to better illustrate the situation.

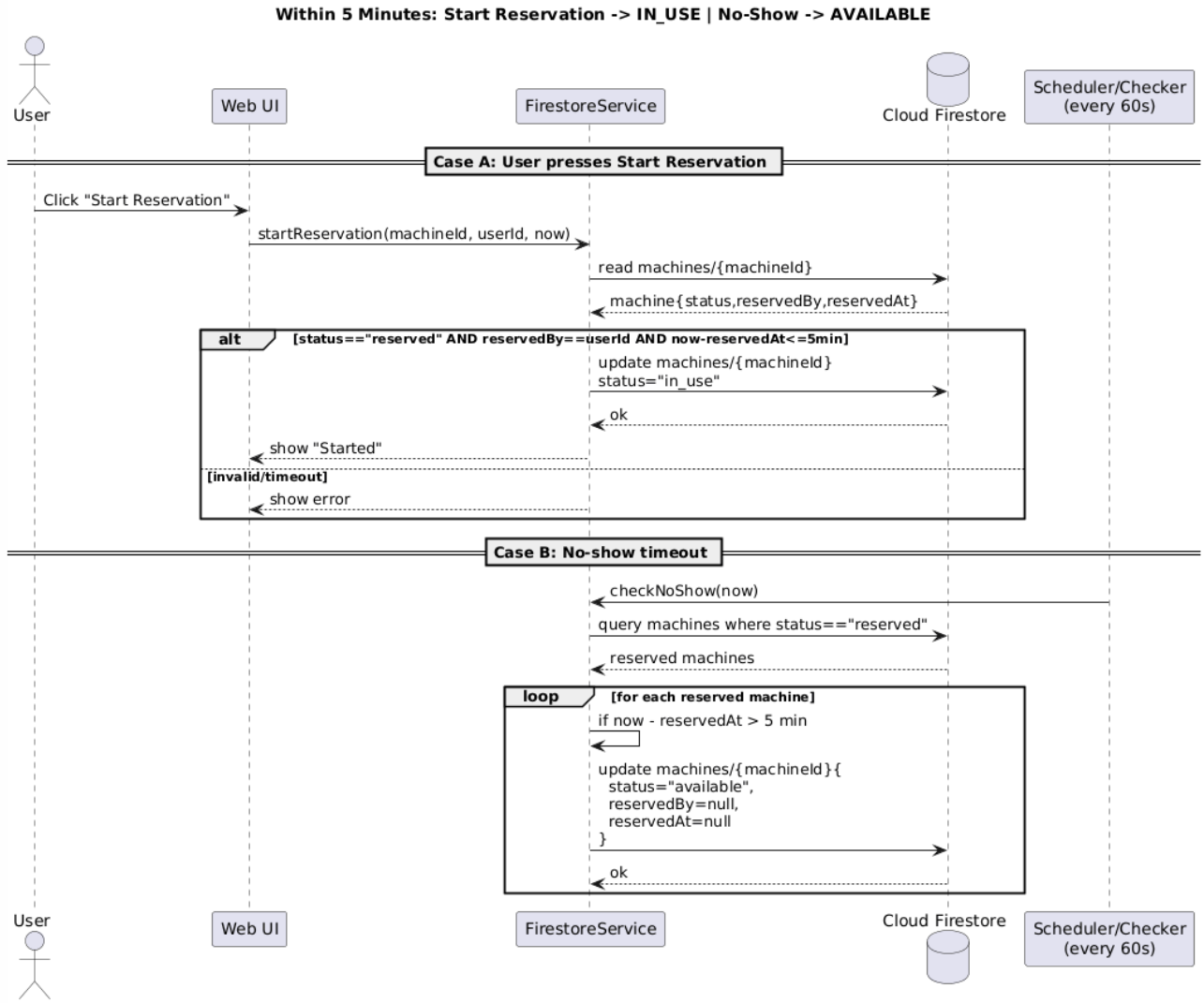


Appointment Time Arrived: Purpose: To automatically set the machine to "reserved" and prepare it to start the appointment when the scheduled time arrives. The Scheduler/Checker running in the background (e.g., every 60 seconds) triggers the checkAppointments(now) function. The system retrieves appointments from the database that fall within the "current time slot". For each appointment found, the data in the Firestore of the relevant machine is updated: status = "reserved" reservedBy = userId.



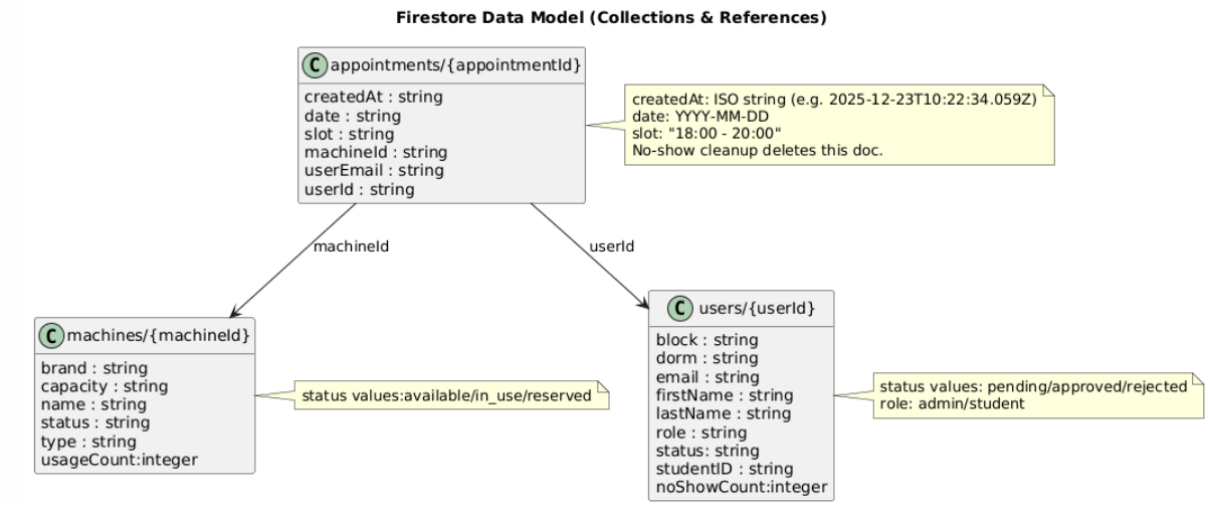
When User Press “Start Reservation” /When user don't press the “Start Now” button:

Purpose: To prove the user actually came, and if they don't, to make the machine available again. The user presses the Start Reservation button within the slot time. The UI calls startReservation(machineId, userId, now). The system reads the machine and verifies it with queries such as: Is the machine truly reserved?, reservedBy this user?, now - reservedAt <= 5 minutes? If the verification is successful, machines.status = "in_use". Thus, all users see that the machine is now in use. This diagram attempts to explain the logic; the naming conventions in the code may vary.



Firestore Data Model

This diagram shows the entities and their fields that we store in Firestore. Firestore stores this data in No-SQL, but we wanted to show our collections and fields together in a sample diagram.

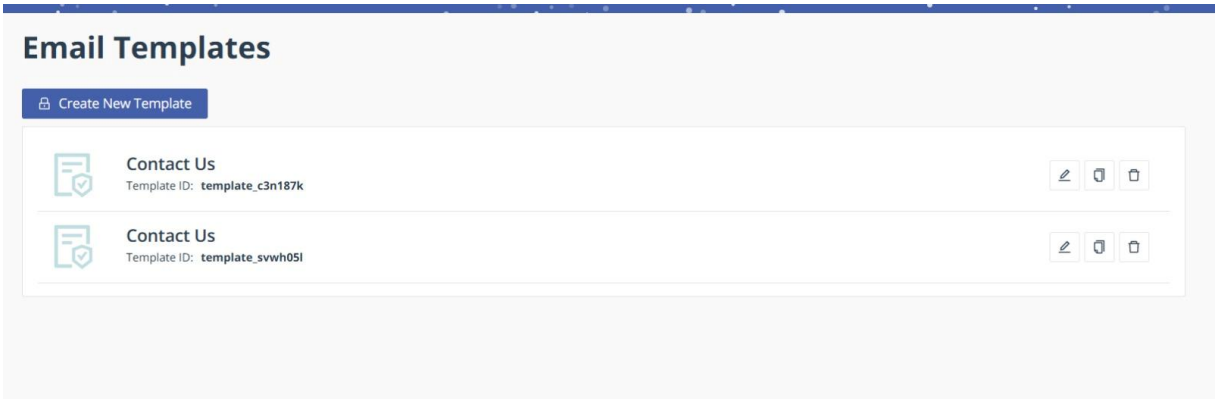


Tutorial for Email.js

Email Provider Setup (EmailJS) – WashMate

To ensure that automated messages appear to actually come from the team, we have established a Gmail account for use in this project (for instance, washmate mailbox).We added this Gmail account to EmailJS and created an “Email Service” for sending emails in the frontend. The Service ID is generated by EmailJS. This is the Service ID that is mentioned inside the JavaScript (.js) file that helps link the app with the Gmail Sending service.

Email Templates (EmailJS)



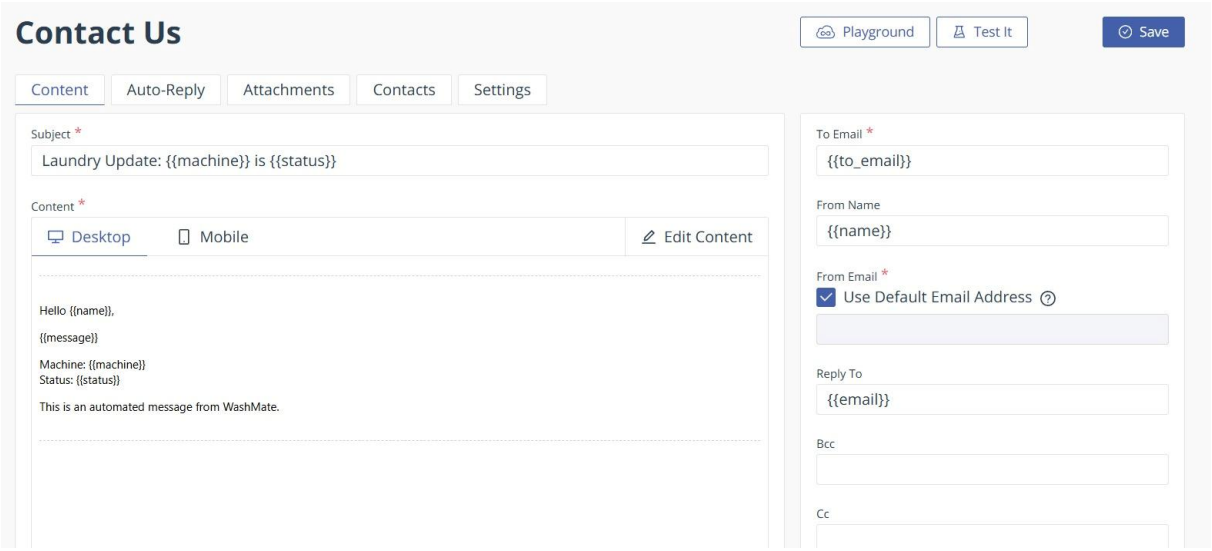
In EmailJS, we designed two email templates to facilitate two primary notification requirements in WashMate:

- Machine_Status_Template
- Launched to update the user regarding the status of laundry machines or booking notifications (e.g., availability/busy).
- Username / Account Template
- Used when personal login and account-related emails are to be sent to users.

Every template has a distinct Template ID, as can be seen on the EmailJS Templates page.

- Within our JavaScript (.js) file, we invoke the EmailJS function by using:
- the Service ID (which email service to use)
- and the proper Template ID according to the message structure.

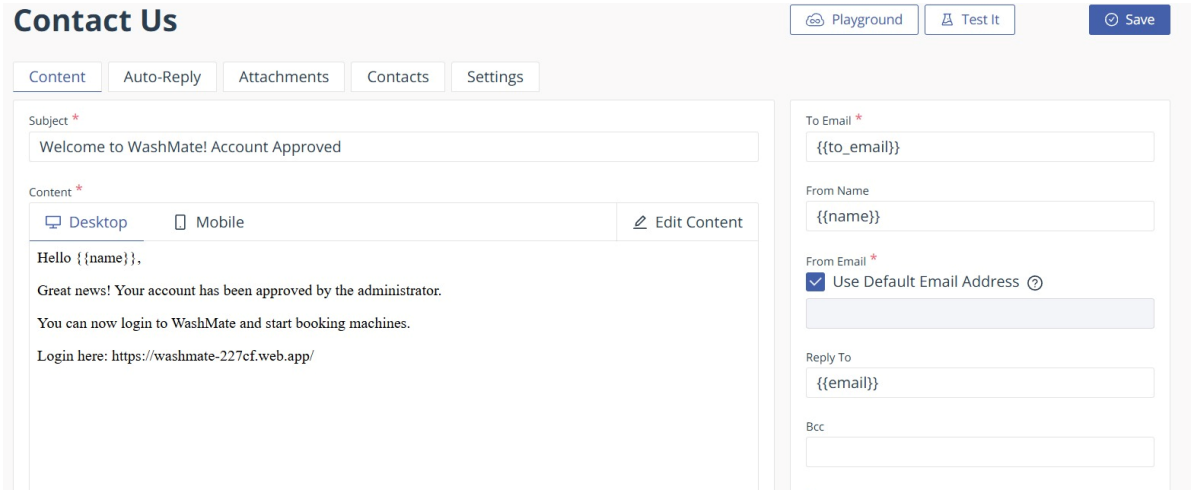
Machine Status Email Template (EmailJS)



This screen shows the **Machine Status email template** created with **EmailJS** for the WashMate system.

- The **email subject** is dynamic and uses variables:
  - {{machine}} → the machine name
  - {{status}} → the current machine status (started/finished)
- The **email content** is also dynamic and personalized:
  - {{name}} → recipient name
  - {{message}} → custom notification message
  - {{machine}} and {{status}} → machine details sent from the system

- The **To Email** field uses {{to_email}}, allowing emails to be sent dynamically to different users.
- The **From Email** address is set to the default WashMate Gmail account configured in EmailJS.
- This template is triggered from the **JavaScript (.js)** file by referencing:
    - the **Service ID** (email service), and
    - the **Template ID** of this email.
- The template is used to automatically notify users whenever a laundry machine's status changes.



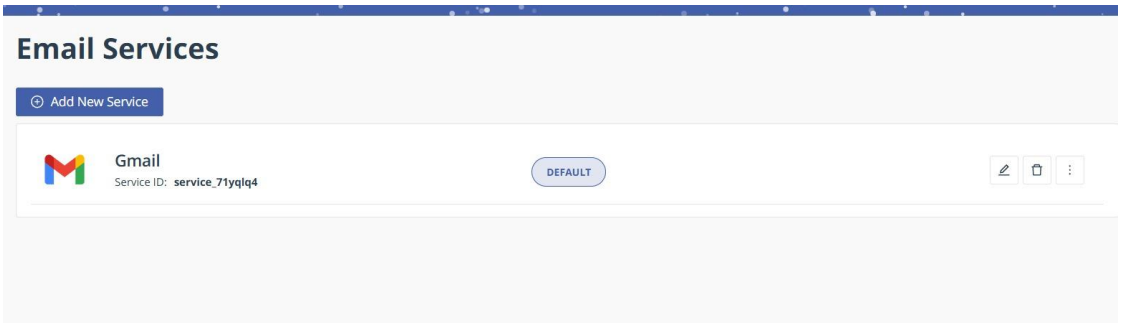
This screen shows the **Account Approval email** that users receive when they are welcomed to the WashMate system. The email template was created using **EmailJS** specifically for the WashMate application. The email subject clearly informs the user about their account status, stating that their WashMate account has been approved. This makes it immediately clear why the email was sent.

The email content is personalized and tailored to each user. The variable {{name}} displays the name of the approved user, ensuring that the message feels direct and user-specific. The message confirms that the administrator has approved the account and that the user is now officially part of the WashMate system.

The email also provides clear instructions on how to get started, including a direct link to log in to the WashMate web application. This allows users to immediately access the system and begin using its features.

The recipient of the email is determined dynamically using the **To Email** field, which references {{to_email}}. This enables the system to automatically send the approval email to the correct user after their account is approved.

All emails are sent from the default WashMate Gmail address configured in EmailJS. This ensures consistency and helps users recognize the emails as official WashMate communications. The template is triggered from the JavaScript (.js) file by referencing the configured **EmailJS Service ID** and the corresponding **Template ID**. This automation allows users to be notified instantly when their WashMate account is approved and ready for use.



This screen shows the **Email Services** configuration page in **EmailJS** used by the WashMate system.

Here, a **Gmail email service** has been created and connected to EmailJS. The service uses a dedicated WashMate Gmail account and is identified by a unique **Service ID** (service_71yqlq4). This Service ID is later referenced in the JavaScript (.js) file to send emails programmatically.

The service is marked as **DEFAULT**, which means all email templates (such as account approval and machine status notifications) will use this Gmail service automatically unless another service is specified. By configuring this email service:

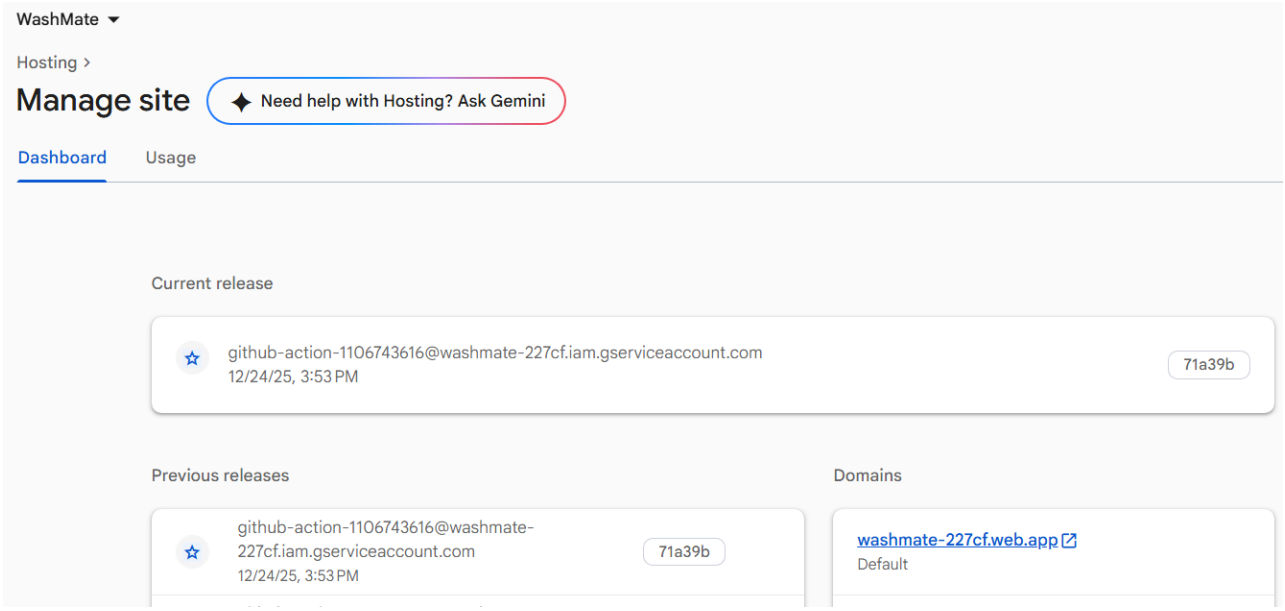
- WashMate can send automated emails directly from its official Gmail address.
- All outgoing emails remain consistent and recognizable to users.
- The backend JavaScript logic can trigger emails reliably using the Service ID without manual intervention.

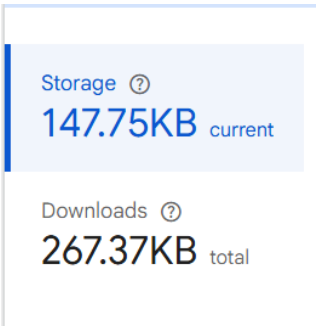
Firebase Changes

We've described Firebase cloud services in detail in the progress report. This section provides the updated version.

Firebase Authentication: No changes were made to this section; only more administrators and users were added to the system.

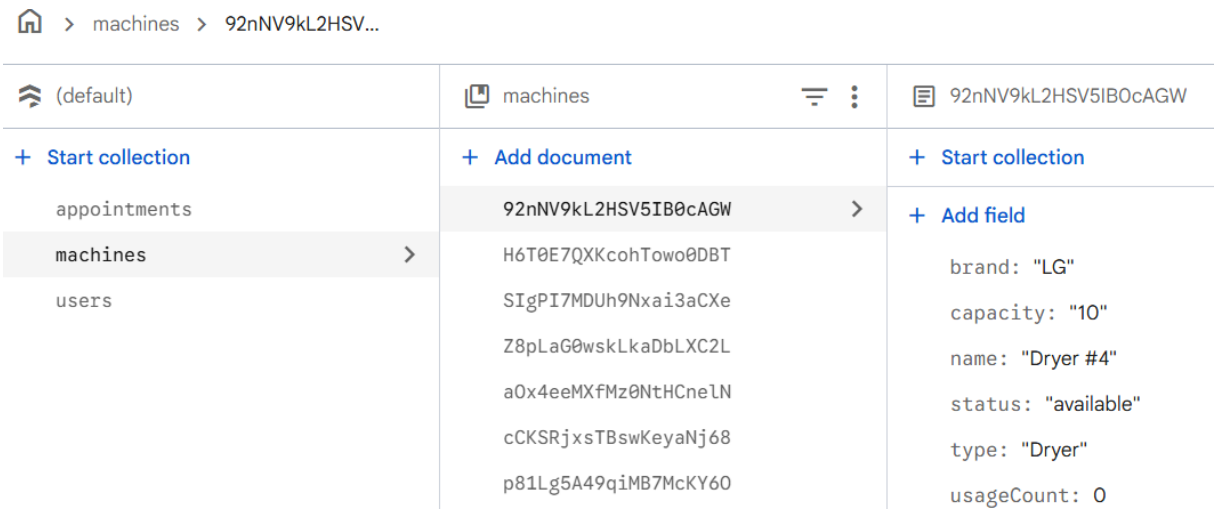
Firebase Hosting: In this section, we track our releases, domains and metrics.

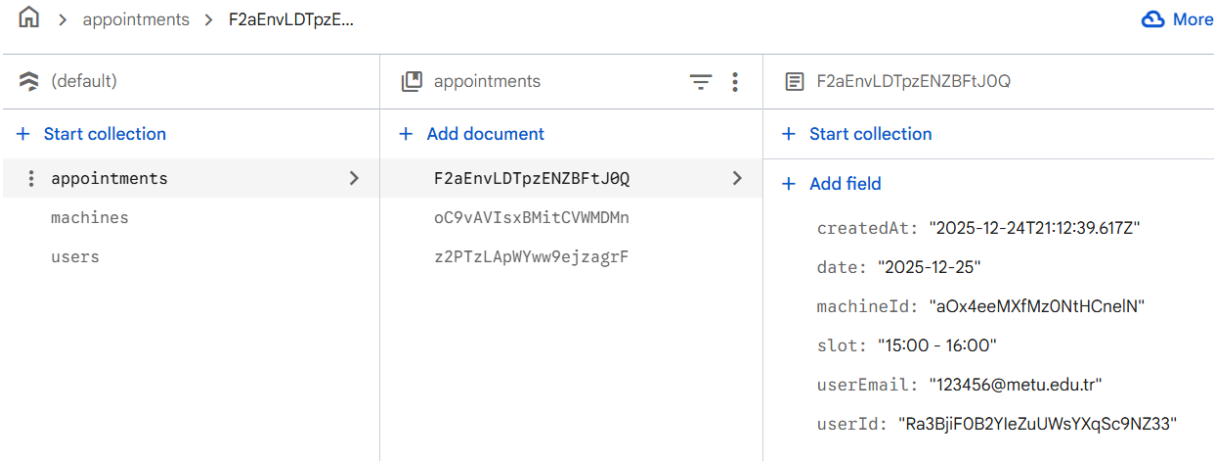


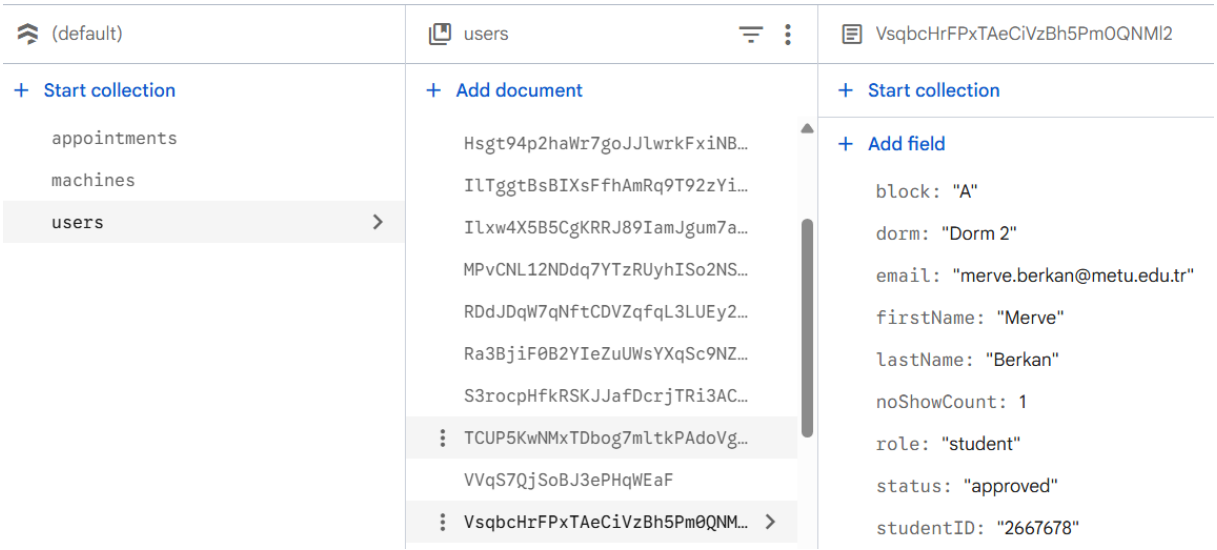


Firestore: This section was modified to accommodate the necessary collections and fields for our data to be suitable for our project. And after the initial developments, the RULE section was also updated to reflect the final version of the project**.**

- Data collections and fields







- Rule: In Firestore, Rules is a server-side structure that controls who can make read/write requests to the database and under what conditions. Our rules are as follows: only logged-in users are allowed to access the Firestore. Users can create and read their own /users/{uid} documents. Admins can read, update, and delete all users. Everyone can read and update machines (such as changing their status), but adding/deleting machines is only for admins. On the Appointments side, anyone logged in can create and read appointments, and anyone logged in can also delete appointments.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    //users
    match /users/{userId} {
      // Allow create if it's their own account
      allow create: if request.auth != null && request.auth.uid == userId;
      // Allow read if it's their own account OR if requester is Admin
      allow read: if request.auth != null && (request.auth.uid == userId || get(/databases/$(da
      //Allow update if it's their own account OR Admin
      allow update: if request.auth != null && (request.auth.uid == userId || get(/databases/$(
      // Only Admin can delete users
      allow delete: if request.auth != null && get(/databases/$(database)/documents/users/$(rec
    }
    //machines
    match /machines/{machineId} {
      allow read: if request.auth != null;
      // Only admins add/remove machines
      allow create, delete: if request.auth != null && get(/databases/$(database)/documents/use
      // Anyone can update status (for Start/Finish/Maintenance)
      allow update: if request.auth != null;
    }
    //appointments
    match /appointments/{appointmentId} {
      // Allow any logged-in user to see slots and book
      allow read, create: if request.auth != null;
      // Allow users to cancel (delete) bookings
      allow delete: if request.auth != null;
    }
  }
}
```

Technologies Used

Programming Languages

- HTML
- CSS
- JavaScript

APIs & SDKs

- Firebase JavaScript SDK
- Firebase Authentication API
- Firebase Firestore API
- EmailJS Browser SDK

Cloud Services

- Firebase Authentication (user login & role management)
- Firebase Firestore (NoSQL, real-time database – DBaaS)
- Firebase Hosting (web application deployment with CDN & HTTPS)
- Email.js (email notifications)

Other Tools

- Git & GitHub (version control and collaboration)

Project Timeline and Responsible Members

Project Part	Description	Timeline	Responsible Member(s)
Project Planning & Proposal	Project idea selection, requirement analysis, proposal writing	Weeks 1–3	Fatih Demirbilek, Nazlıcan Taviş, Nisa Sağdıç
Cloud Technology Research	Research and comparison of AWS and Firebase services	Weeks 4–5	Fatih Demirbilek, Nazlıcan Taviş, Nisa Sağdıç
Client-Side Web Application	Frontend development using HTML, CSS, and JavaScript	Weeks 6–9	Fatih Demirbilek, Nisa Sağdıç
Authentication Component	Student/admin login and registration with Firebase Authentication	Weeks 7–8	Fatih Demirbilek, Nisa Sağdıç
Database Design (Firestore)	Firestore collections and data modeling	Weeks 7–8	Fatih Demirbilek, Nazlıcan Taviş Nisa Sağdıç
Admin Dashboard	User approval, machine add/remove, statistics view	Weeks 8–9	Nazlıcan Taviş
Booking & Scheduling System	Reservation logic, conflict prevention, no shown rules	Week 11	Nisa Sağdıç
Notification & Automation	Email notifications and warning mechanisms via Email.js	Week 12	Fatih Demirbilek Nazlıcan Taviş
Analytics & Advanced Admin Features	Advanced management operations, Machine usage statistics and information pages	Week 12	Nazlıcan Taviş
Testing & Validation	Test case creation, system testing, bug fixing	Week 13	Fatih Demirbilek, Nazlıcan Taviş, Nisa Sağdıç
Final Report & Demo	Final documentation and project presentation	Week 14	Fatih Demirbilek, Nazlıcan Taviş, Nisa Sağdıç

Metrics

Lines of Codes

- App.js: 1030 line
- Index.html: 496 line
- 404.html: 33 line

Programming Lanuages

- Html
- CSS
- JavaScript

Memory Requirements

1) Client (user's computer)

The application runs in a browser; memory usage depends on the browser and open tabs.

Minimum: Works with a modern browser on a device with 2 GB of RAM.

Recommended: 4 GB+ RAM (for a UI + Firestore real-time updates).

2) Server/Backend

Firebase managed services are being used. Therefore "server memory requirement" = N/A

3) Project file in our computers

The entire Washmate project takes up 96MB of space on the disk.

Data Types

We use Firebase Cloud Firestore, a NoSQL document database (DBaaS). Data is stored in collections (users, machines, appointments) as JSON-like documents containing primitive field types such as strings, numbers. A detailed representation can be found in the FireStore DataModel table.

Github

You can reach our repository from the given link below. The repository is public and available: **https://github.com/NazlicanTavis/WashMate.git**

Also you can reach our websie via using link: **https://washmate-227cf.web.app**

References

- *draw.io - free flowchart maker and diagrams online*. (n.d.). https://app.diagrams.net/
- Emilieanthony. (n.d.). *GitHub - emilieanthony/laundry-booking: Full-stack laundry booking system with a client-server architecture built in Vue, Node and Express. Project in DIT341 Mobile- and Web Development.* GitHub. https://github.com/emilieanthony/laundry-booking
- *Firebase CLI*. (n.d.). Firebase Open Source. https://firebaseopensource.com/projects/firebase/firebase-tools
- *Firestore | Firebase*. (n.d.). Firebase. https://firebase.google.com/docs/firestore
- Lundsnation. (n.d.). *GitHub - lundsnation/laundry_booking: A simple laundry booking system built with Next.js, mongoDB and auth0.* GitHub. https://github.com/lundsnation/laundry_booking
- Tkyk. (n.d.). *GitHub - TKYK93/Laundry-booking*. GitHub. https://github.com/TKYK93/Laundry-booking
- *Tutorial | EmailJS*. (n.d.). EmailJS. https://www.emailjs.com/docs/tutorial/overview/