Habitat Connect

Laura Rodríguez López
Department of Computer Science
Hanyang University
Madrid, Spain
1postlaura@gmail.com

Christian Alexander Hernandez
Department of Computer Science
Hanyang University
Dallas, United States
ch407541@gmail.com

Marcel Plesky
Department of Information Systems
Hanyang University
Zürich, Switzerland
marcel.plesky68@gmail.com

***Abstract*—Habitat Connect is a web application designed for residents of Hanyang Habitat, our apartment complex, to stay connected, regardless of their cultural backgrounds. Our platform offers features like an announcement board, birthday calendar, washer checker, and maintenance request page. Residents can easily receive important updates, plan events, check laundry availability, and report maintenance concerns – all within a user-friendly interface. With Habitat Connect, we hope to foster a stronger sense of community and make everyday tasks more convenient for everyone in Hanyang Habitat.**

ROLE ASSIGNMENTS

In the table below, we can find a detailed breakdown of the roles and responsibilities assigned to each team member throughout the duration of the project.

Role	Name	Task Description
User/Customer	Laura	- Uses the software - Establishes the conditions and features that the software must satisfy (requirements) - Provides feedback - Approves change requests
Software Developer	Christian	- Works with user/customer to gather their needs - Designs and develops the software - Tests and debugs the software
Development Manager	Marcel	- Supervises the software development process - Facilitates communication between user/customer and software developer - Manages tools and resources - Monitors progress and quality of the software

TABLE I: Roles and Responsibilities

I. INTRODUCTION

A. Motivation

When international students arrive at Hanyang University, they often seek affordable and short-term housing options. One of the most popular accommodations are Goshiwons like Hanyang Habitat. As members of this team currently living in Habitat, we would like to provide residents with a new toll that caters to their needs.

Living in Hanyang Habitat has given us firsthand insight into the challenges faced by exchange students. With approximately 70 international residents that come from more than 15 countries, it's evident that online communication platforms vary widely. For instance, Europeans prefer applications like WhatsApp or Instagram, while North Americans rely heavily on text messaging. In contrast, WeChat is very popular in certain Asian countries like China or Indonesia. In South Korea, KakaoTalk is the go-to messaging app, but its unique features and interface can pose a learning curve for newcomers, especially for those that do not speak Korean.

B. Problem Statement

Given these cultural differences regarding communication tools, we asked ourselves: which platform should Hanyang Habitat residents use to effectively communicate with one another? Our proposed solution is Habitat Connect, an announcement board with a user-friendly English interface.

As well as serving as an online communication tool, Habitat Connect offers more features that focus on the needs of Habitat residents. This includes a laundry checker that provides real-time updates on washer availability, a birthday calendar to keep everyone informed about upcoming celebrations, and an easy-to-use maintenance request system, ensuring a direct line of communication between the tenants and the apartment manager.

Habitat Connect ensures that all residents, regardless of their language and online backgrounds, can comfortably navigate and utilize the tool. This idea comes from a desire to foster a sense of community among students in Hanyang Habitat. We aim to enhance the living experience of international students, making their time at Hanyang University more comfortable and rewarding.

C. Research on related software

WHATSAPP

WhatsApp is a widely used messaging application that offers instant messaging, voice and video calls, file-sharing and even a poll-system that allows voting for user given options. WhatsApp is used in family, friends and even work environments to stay connected and up to date with your contacts.

KAKAOTALK

Just like WhatsApp, KakaoTalk is a popular messaging application. It offers even more features than WhatsApp like mobile payment and other various app services making it a multifunctional communication platform.

DISCORD

Discord is known for its features like text and voice chat, creating and managing channels, and building communities around specific interests or topics. It's often used for both casual and structured discussions, making it a versatile option for promoting communication and collaboration in a variety of settings, including apartment buildings and shared living spaces.

GOOGLE CALENDAR

Google Calendar is a user-friendly online calendar application from Google. It allows users to create, manage and share events, streamlining planning. Moreover, users can also import birthdays from their contacts, making it easier to remember important dates. Google integration and various viewing options make it a valuable time management and organization tool.

FAMILYWALL

FamilyWall is an application designed to help families and individuals organize their lives, share information, and manage family-related activities. It allows users to create shared calendars, to-do lists, shopping lists, and share photos, as well as important information with family members or within their respective groups.

LAUNDRY APPLICATIONS

Some laundromats like Samsung and LG have developed their own mobile apps to check the availability of washing machines and dryers.

ZENDESK

Zendesk is a popular ticket management system used for customer support and issue tracking. It helps organize and track customer inquiries and support tickets. With Zendesk, businesses can provide better customer service.

II. REQUIREMENT ANALYSIS

A. Register via Local Registration

If logged out, or a user is entering the website for the first time, the first component that users will see is the login page, as the website is a private application not open for public use. This page will have two main functions to enter the main webpage, sign up and login. The user sign up process via local registration on Habitat Connect is the starting point for users to establish their digital presence. It allows them to create unique usernames, set up secure passwords, and provide email addresses. These steps collectively enable users to register and create a personalized digital identity within the platform, enhancing their overall user experience.

B. Log in via Local Registration

Upon entering Habitat Connect, users will notice a login button. Clicking on this button redirects the user to the login page, where Habitat Connects offers support for both local and Google OAuth login processes. If the user chooses to login locally, they will be asked for an email address and password. Then, the system conducts an internal search for the user in the database. If the user is found, they will be seamlessly redirected to the dashboard page of Habitat Connect. On the other hand, if the user credentials are not registered in the system's database, the user will be prompted to provide alternative credentials.

C. Register via Google OAuth

Habitat Connect supports both local and GoogleOAuth registration processes. When users choose to register with Google, they engage in Google OAuth, a secure process simplifying registration. This method lets users authorize third-party apps like Habitat Connect, using Google credentials without revealing their password. Our system creates the user's account in our database without further validation, leveraging Google's checks. Once successfully registered, the next step of the registration process redirects users to the complete profile page, where they complete their profile setup. From here, the process is the same as for the local registration.

D. Log in via Google OAuth

Logging into Habitat Connect with Google OAuth offers users a host of advantages, with convenience, security, and efficient access management being chief among them. This approach eliminates the cumbersome need to remember and juggle numerous usernames and passwords for different online platforms, simplifying the user experience significantly. Furthermore, users maintain a high level of control over their digital footprint, with the ability to easily revoke access to specific services through their Google account settings whenever they deem it necessary. This synergy between ease of use and robust data security underscores the appeal and practicality of logging in via Google OAuth. Once Google successfully completes the login validation, users are seamlessly redirected to the main announcement page of Habitat Connect.

E. Logged In Restriction

At Habitat Connect, we prioritize user privacy and security, and that's why we require users to register and log in before accessing the full range of features on our website. Our welcome screen serves as a safeguard, ensuring that only registered users have access to sensitive and personal information such as announcements, the laundry checker, the birthday calendar, and the maintenance request creation feature. By implementing this registration and login process, we not only protect our users' data but also create a more personalized and secure experience for each tenant within our community. This approach allows us to maintain a trusted and respectful online environment where apartment tenants can engage with the platform's various functionalities while respecting the privacy and safety of their fellow users.

F. Profile

The Profile feature on Habitat Connect is an essential component of our platform, providing users with the ability to personalize and update their user information. It addresses the common scenario where users might have entered incorrect details during the sign-up process or experienced changes in their living situation. With the update profile feature, users can easily rectify errors in their usernames and birthdays, ensuring that their accounts accurately reflect their identities and important personal information. This flexibility on the user's account fosters a sense of ownership and control over their information. Additionally, users can update their room numbers, which is especially valuable in apartment complexes where room assignments may change over time due to leasing arrangements or other factors.

In the future, there may be additional features, such as editing the profile picture or setting a public status for every user to see, that will allow users to further express their personality in their Habitat Connect's account.

G. Dashboard

Habitat Connect offers a range of communication and interaction features for its users. The main requirement of our system is to provide users with a platform for communication, focusing on a common platform that users can engage with regarding their cultural background and online habits. For this purpose, we created the dashboard page. On this page, users have the ability to post announcements, which include text, making information accessible to all registered users and ensuring transparency through clear ownership attribution. Users can also edit their announcements to keep them up to date and relevant, and they have the option to delete announcements if they become outdated or if an event was canceled. As users regularly engage with Habitat Connect, reading older announcements may become challenging. To address this, we implemented a search function, allowing users to locate a specific announcement containing the chosen keyword. Additionally, to enhance user interactivity with the dashboard page, users can engage in discussions by commenting on announcements, with comment ownership clearly visible, promoting accountability and a sense of community within the platform.

H. Calendar view

The application's calendar view plays a central role in the user experience, offering both visual appeal and functional benefits. It automatically showcases birthdays, leveraging user sign-up details to populate the calendar. This feature enhances user connection by facilitating easy viewing and remembrance of community members' birthdays. The monthly view and intuitive navigation arrows make the calendar user-friendly and organized. Beyond its practical aspects, the calendar fosters community building by encouraging users to celebrate each other's birthdays, strengthening a sense of togetherness and shared experiences.

I. Laundry checker

The Laundry Checker function is a valuable tool designed to enhance the laundry experience within our apartment complex. It streamlines and optimizes the laundry process by empowering residents to make informed decisions about when to use the machines. With real-time information on washer availability, residents can avoid unnecessary trips to the laundry room during peak usage times, enhancing both convenience and efficiency.

In the future, there may be additional features such as timers to display how long a washer has been in use, as well as user-friendly tooltips or information icons that can provide additional information or instructions.

J. Maintenance requests

The Maintenance Request feature is designed to facilitate and improve the process of reporting and resolving maintenance issues within the apartment complex. This function is indispensable for several compelling reasons, contributing to the overall well-being and satisfaction of our residents. When residents encounter maintenance issues in their apartments, it's crucial to provide them with a user-friendly and efficient means of reporting these problems. Users can easily review and manage their open requests, including updating the title and description as needed, ensuring effective communication with the maintenance team. Habitat Connect not only fosters efficiency but also enhances transparency by allowing users to track the status and progress of their requests.

III. Development environment

A. Choice of software development platform

Given that Habitat Connect is a web application, the choice of operating system is quite flexible. That is, we can choose any operating system that supports web development tools. To be more specific, our working environment consists of the following:

Name	Version	Description
Windows	11 Home	- User-friendly environment with an updated UI - Wide range of web development tools and IDEs
macOS	12.5 Apple M1 Pro	- Unix-based - Offers a powerful terminal for web development

TABLE II: Working environment

Moreover, our development process will make use of some tools that can be found below.

1) Node.js is a server-side runtime environment for executing JavaScript code, enabling efficient web application development. It excels in handling real-time interactions and concurrent requests.

Fig. 1: Node.js logo

2) EXPRESS.JS is a popular and minimalist web application framework for Node.js. It provides a robust set of features and tools for building web and mobile applications, making it one of the most widely used frameworks in the Node.js ecosystem.

Fig. 2: Express.js logo

3) PASSPORT is a widely-used authentication middleware for Node.js applications. It simplifies the process of implementing user authentication in web applications by providing a flexible and modular framework.

Fig. 3: Passport logo

4) BOOTSTRAP is a front-end framework that provides predesigned, responsive, and customizable UI components. It allows us to create appealing and web applications with ease.

Fig. 4: Bootstrap logo

5) MONGODB is a NoSQL database that provides a flexible, document-based data structure. It's designed for handling large amounts of unstructured data, making it suitable for dynamic web applications.

Fig. 5: MongoDB logo

6) VISUAL STUDIO CODE is a powerful code editor that supports various programming languages. Furthermore, this IDE offers features like syntax highlighting, debugging, extensions, and Git integration.

Fig. 6: Visual Studio Code logo

7) GIT/GITHUB: Git is a version control system that allows multiple developers to collaborate on a project. On the other hand, GitHub is a web-based platform that provides Git repositories, collaboration tools, and hosting services. It facilitates efficient team collaboration and code management.

(a) Git logo

(b) GitHub logo

Fig. 7: Logos for Git and GitHub

8) GOOGLE DRIVE is a cloud-based storage and collaboration platform. It allows us to store files, share documents, and collaborate in real-time. It's useful for storing mock documents that will be pushed to GitHub.

Fig. 8: Google Drive logo

9) GOOGLE OAUTH is a secure and widely-used authentication protocol that allows users to log in to various websites and applications using their Google credentials, eliminating the need for creating and remembering separate usernames and passwords.

Fig. 9: Google OAuth logo

10) OVERLEAF is an online platform for collaborative document editing. It simplifies the process of creating and editing scientific and technical documents, so it's very

popular among researchers and academics. We will make use of it to create our LaTeX documents.

Fig. 10: Overleaf logo

11) MOQUPS is a web-based design and prototyping tool that allows users to collaborate on UI design in real-time

Fig. 11: Moqups logo

Regarding the choice of programming language, we will be using the following:

1) JAVASCRIPT is a high-level programming language primarily known for its use in web development. It allows us to add interactivity and dynamic behavior to web pages. With Node.js, it can also be used for server-side development.

Fig. 12: JavaScript logo

2) HTML is the standard language used for creating web pages. It provides the structure and content of a webpage, using elements like headings, paragraphs, and links.

Fig. 13: HTML logo

3) CSS is a language used for styling web pages. It controls the visual presentation of HTML elements, including aspects like layout, colors, fonts, and animations.

Fig. 14: CSS logo

All software development tools are free of use. Furthermore, since Windows11 and macOS are already in use, we will not count them as costs.

B. Software in use

Today people rely on widely-used software like Discord for chatting, Google Calendar for planning, and FamilyWall for staying organized. Our upcoming project takes inspiration from these popular tools and blends them into one. Our software will make it easy for people living in the same place to chat, schedule events, and share information, enhancing the way they interact and stay connected.

C. Task distribution

In such a collaborative project among three students, roles have been thoughtfully distributed to capitalize on our individual strengths. We believe that not dividing the tasks will result in an inefficient and not realistic way of software development. The workload can be divided into distinct categories, each focusing on a crucial aspect of Habitat Connect. The tasks are as follows:

- **Management**: responsible for checking up on each team member work, ensuring adherence to deadlines, and coordinating the overall workflow
- **Documentation**: responsible for writing the pdf and creating the LaTex files, ensuring to present a well-structured and comprehensive overview of the project
- **Frontend**: charged with the responsibility of developing the frontend side of the software. This includes crafting the user interface, ensuring a seamless and interactive user experience, and design and implement every component for user engagement
- **Backend**: charged with the responsibility of developing the backend side of the software. This includes handling the database, server-side logic, and ensuring that every software component correctly adheres to all specified requirements

After thoughtful evaluation, the final assignment of tasks to each team member has been documented in the following table:

Name	Role	Responsibilities
Laura	Backend Frontend Management Documentation	As the backend developer, this member is responsible for constructing the software foundation code. This involves turning our project requirements into code, integrating external libraries to enhance our project capabilities, and addressing bugs to maintain a robust and functional backend infrastructure. As frontend developer, this member undertook the responsibility of designing and crafting a basic frontend infrastructure, providing a solid starting point for the rest of the team. Moreover, this member's management tasks included taking responsibility for overseeing the team dynamics and workflow. Not only involves a checking up on each team member's work but also ensuring strict adherence to deadlines. Additionally, as a documentation assistant, this member was charged with the task of reviewing documentation for clarity, suggesting ideas for improvement and organization, and actively participating in the creation of LaTex files.
Marcel	Frontend Documentation	As a frontend developer, this member assumed the responsibility of reshaping the user interaction within the software system. This involves not only the rearrangement of Bootstrap elements but also the incorporation of fresh and engaging content that contributed to an enhanced and visually appealing user experience. This member approach was not merely aesthetic but strategic, aiming to create an immersive and visually appealing environment that resonated with end users. Moreover, this member's tasks also included an active participation in the documentation tasks, providing critical assistance to the main documentation writer.
Christian	Frontend Documentation	In his role as a frontend developer, this member's tasks included refining the user interface by further rearranging bootstrap elements. As the main documentation writer, he was entrusted with documentation responsibilities, ensuring that all elements are constantly updated and aligned with code changes. This approach to documentation ensures that the project's report remains accurate and reflective of the software's evolving codebase.

TABLE III: Task Distribution

IV. Specifications

In the section below, we will provide a detailed description of the implementation process of each requirement.

A. Register via Local Registration

To locally register in Habitat Connect, users need to provide a valid email address and a password. They must comply with the following rules:

- Email: users must provide a valid email address. If not, a registration fail message will be shown
- Password: string of at least 8 characters. Must contain at least one capital letter

If the user fills the form with invalid inputs, a validation error message will appear on the screen. Once all inputs are valid and the user clicks the register, we create an instance of the user in our MongoDB database. From the users point of view, they will be transported to the complete profile page to input more essential information to complete their registration process.

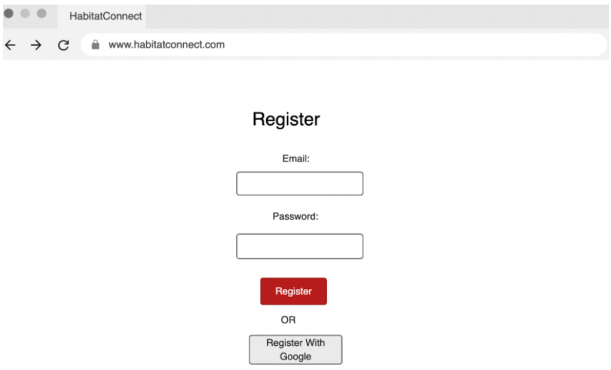


Fig. 15: Registration page (mock up)

B. Log In via Local Registration

To log in locally, users must provide their email and password. If the email and password of the user matches the information in the MongoDB database in regards to our local registration, the user will be successfully transported to the announcement page. If, on the other hand, the information doesn't match, the user will get a validation error on the screen. It must be noted that users cannot access any Habitat Connect main pages, such as the dashboard or laundry checker pages, until the login process has been successfully completed.

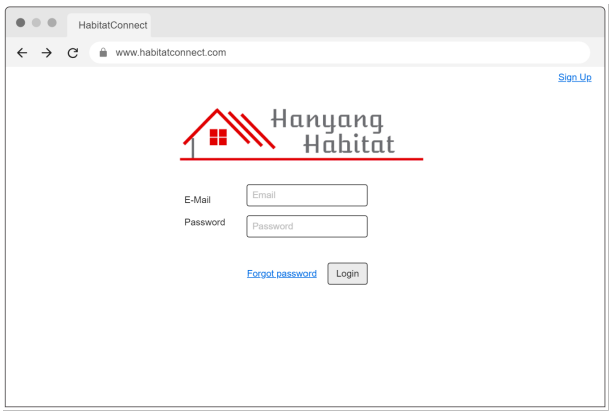


Fig. 16: Log In page (mock up)

C. Register via Google OAuth

Users initiate this process by clicking the register with Google button on Habitat Connect's registration page. To create an instance of the user in our system's database, users will need to select a Google account and login. Regarding the user's privacy, Google will show the requested permissions that Habitat Connect will make use of. Users have the option to approve or reject the use of their personal information. However, if not given the permission for Habitat Connect, the registration process will not be completed and users will be redirected to the homepage. If, on the other hand, the user

grants the permissions needed, Google generates an OAuth token, confirming identity and consent. Our system creates the user's account in our database without further validation, leveraging Google's checks. It is important to note that when users register via Google OAuth, their Habitat Connect account lacks a password field. As a result, users who register with Google OAuth can only login using Google. The next step of the registration process redirects users to the complete profile page, where they complete their profile setup. From here, the process is the same as for the local registration.

D. Log In via Google OAuth

When users choose to sign in using Google OAuth, the application redirects them to Google's authentication system, where users can then choose the account they want to login with. Similarly to the Google OAuth registration process, Google performs the validation of the account. Once everything is on order, the user will be transported to the dashboard of Habitat Connect. However, if a login error arises, users will be redirected to the homepage and they will not be able to access Habitat Connect's main functionalities.

E. Logged In Restriction

To protect Habitat Connect's main features and users' personal information, we require users to login before accessing pages like the dashboard or the laundry checker. Everytime the user attempts to access a protected page, the system performs an internal checking and determines whether the user has a current session opened. If the session does not exist yet, it means that the user has not logged in. As a result, users will be redirected to a new page where they will see the corresponding validation error message (Sorry, you are not authorized to access this resource. Please make sure you have the necessary credentials). If, on the other hand, the user has logged in, the system will find an open session and will not trigger the unauthorized page. As a result, logged in users won't encounter any problems when accessing Habitat Connect's protected pages. The protected pages are: profile, dashboard, calendar, laundry checker, and maintenance requests.

F. Profile

The first interaction that users have with their profile is the complete profile page. This page is only visited after the registration process to complete the missing fields in the user's account in the MongoDB database. Otherwise, the user has no direct access to this page. Regarding the missing fields, we expect the following inputs:

- `Username`: string of maximum 12 characters. Moreover, must be a unique value in our database
- `Birthdate`: date in MM/DD/YYYY format. Users have to input their birthdate with numbers on their keyboard, or choose a date from the birthday form
- `Room number`: number between 300 and 500. Moreover, must be a unique value in our database

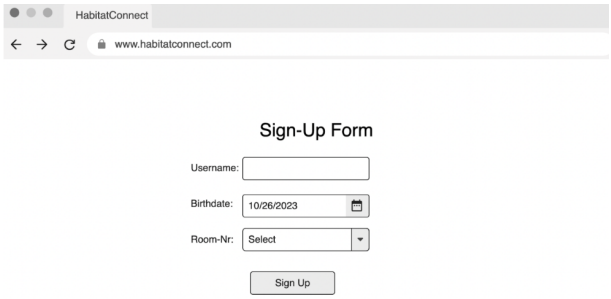


Fig. 17: Sign up form (mock up)

If the user fills the form with invalid inputs, a validation error message will appear on the screen. Once all inputs are valid and the user clicks the complete profile button, they will be redirected to the dashboard page. On the system's side, we will create an instance of the user in the database.

Moreover, users can access their profile information by clicking on the corresponding button in the navigation bar. In the user profile section, users have the flexibility to make adjustments to several key aspects of their accounts to ensure a personalized experience.

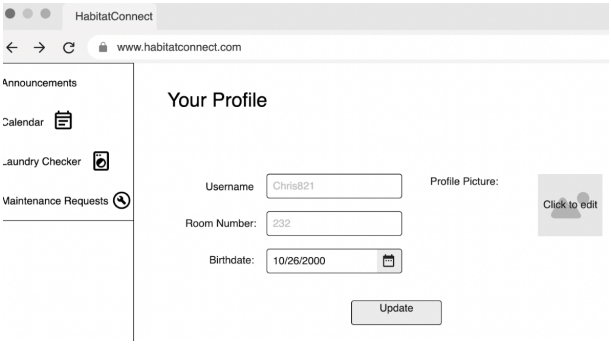


Fig. 18: Profile page (mock up)

Users can change their room number, birthdate, and username as many times as desired. The corresponding validations, previously mentioned in the complete profile page, will be applied to avoid the user submitting invalid values.

G. Dashboard

The announcement board will be the first thing that users will see upon logging into our web application. The user is able to see all announcements ever posted, except those deleted by their corresponding authors.

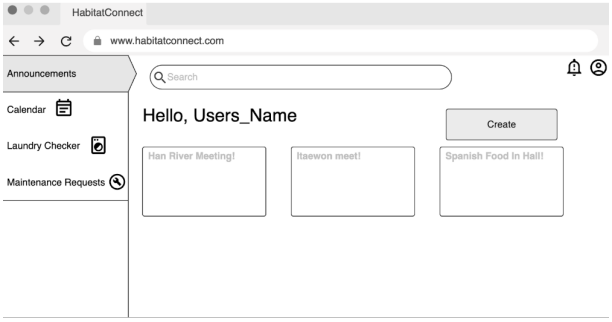


Fig. 19: Dashboard page (mock up)

In the dashboard page, the user can perform the following actions:

- **Post** an announcement: users can write announcements that can only contain ASCII characters. Once they choose to post the announcement, it will be available in the dashboard to every user registered in Habitat Connect. The ownership of the announcement is registered as well. An announcement has the following fields:
  - `Title`: string of maximum 500 characters
  - `Body`: string of maximum 1,000 characters

  In case of submitting invalid fields, the user will see the corresponding validation error message.

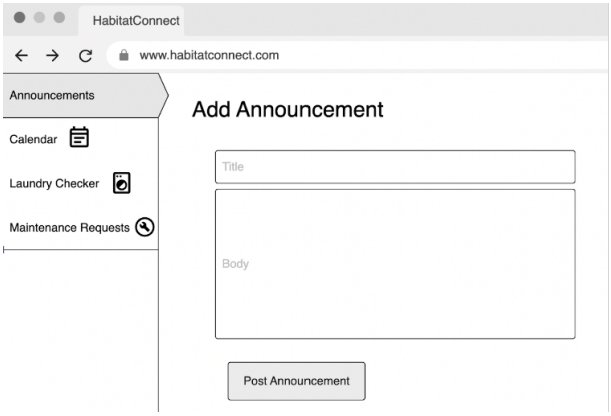


Fig. 20: Add announcement page (mock up)

- **Edit** an announcement: users can edit the title and/or body of their posted announcements. To edit a post, users must click their own announcements, where only the owner will see an update button. Regarding the inputs expected, they follow the same validation process for posting an announcement. That is, the title must not be longer than 500 characters, and the body must be at maximum 1,000 characters. Once finished with their edited announcement, they can click the update button in red under the body. The user will be redirected to the dashboard and they will see their updated announcement at the top of the dashboard.

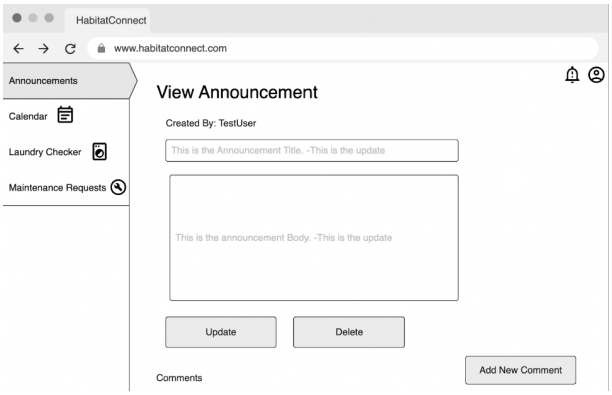


Fig. 21: View announcement page (mock up)

- **Delete** an announcement: users are able to delete their posted announcements. They will no longer appear in the dashboard, and they will be removed from the system's database. Once the user clicks on the delete button, they will see a confirmation popup saying *Are you sure you want to delete this announcement?*, with a cancel/delete option. If the delete action is confirmed, the announcement will be deleted immediately, and the announcement below it will move up one. This action can only be done by users who created their announcement (ownership privilege). That is, no user can delete another user's announcement.

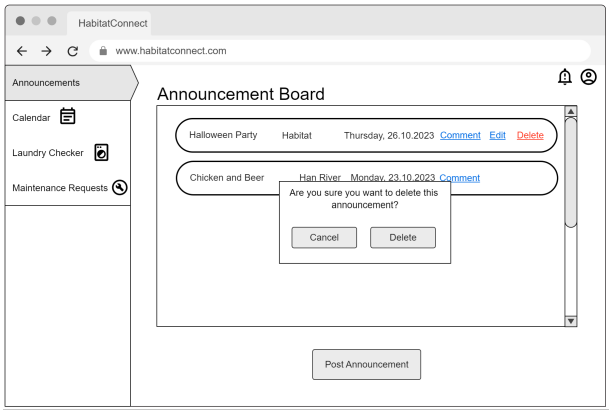


Fig. 22: Delete announcement confirmation popup (mockup)

- **Comment** on an announcement: registered users can comment on posted announcements. Comments can only contain ASCII characters, and must not be longer than 500 characters. If the user doesn't follow the validation, they will see the corresponding error message. Furthermore, the ownership of the comment is available for every user to see.

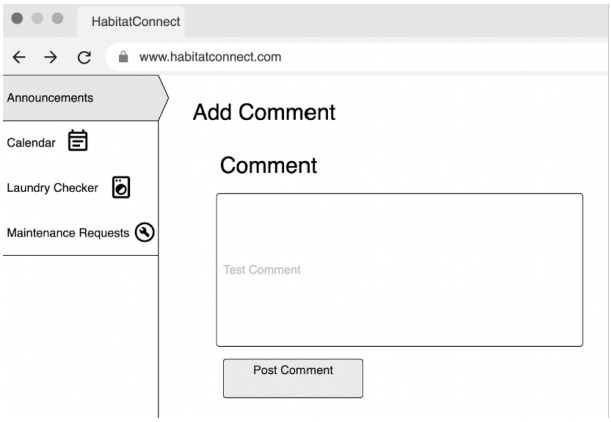


Fig. 23: Add comment page (mock up)

- **Edit** a comment: users can edit their posted comments. They must comply with the same validation rules as when posting a comment. That is, the comment length cannot be larger than 500 characters.

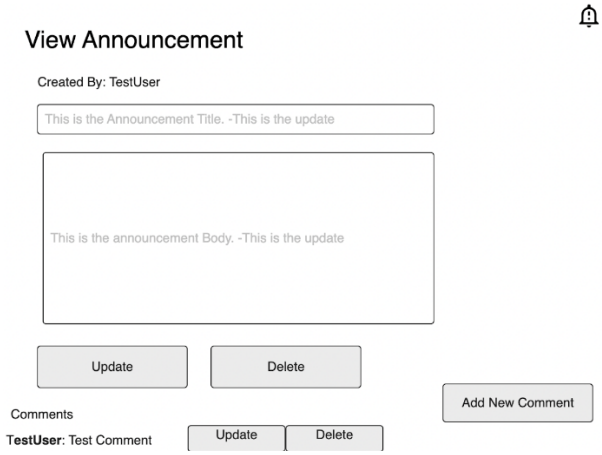


Fig. 24: Edit comment option (mock up)

- **Delete** a comment: users are able to delete their posted comments. They will no longer appear as comments in the announcement view, and they will be removed from the system's database.
- **Search** an announcement: users can search for announcements by entering keywords, which are then searched in both the title and body of the announcement. At the top of the announcement board page, there is a search bar that will be visible. This function is available to all users regardless of ownership of the post. The sorting of the announcements are in ordered form from latest to oldest, there is an option to go to the next page or previous page.

H. Calendar view

The calendar view can be directly accessed through the task bar on the left side of the screen. When users first open the calendar page, it automatically displays the current month, thanks to the initialization of the currentDate object in the code, which is designed to capture the current day, month, and year. Furthermore, the calendar is populated by taking the user's birthday field stored in our MongoDB database.

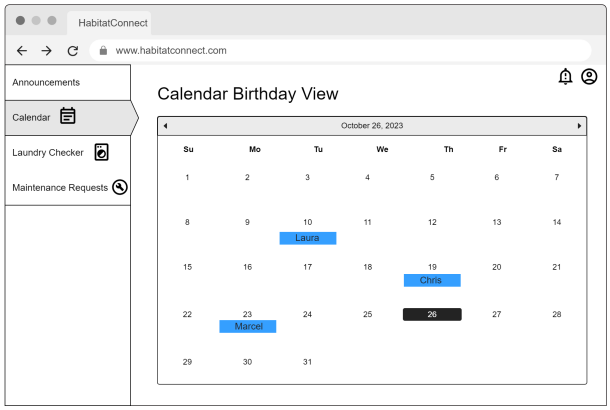


Fig. 25: Calendar view page (mockup)

This page will not have any actions, and is more of a visual aid for birthdays. The month switch function in the code allows users to navigate between different months displayed in the calendar. When users click on the previous month or next month buttons, the JavaScript function is triggered. For instance, if the user clicks the previous month button, the function checks the current month and year displayed on the calendar. If it's January, the function resets the month to December of the previous year; otherwise, it simply decrements the month. Conversely, the same logic applies when clicking on the next month button.

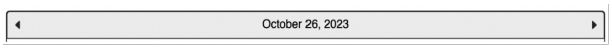


Fig. 26: Calendar navigation option (mock up)

I. Laundry checker

To access the laundry checker page, the user must click on the corresponding button in the navigation bar. In this page, users will see the washing machines information in a table-like format, ensuring that residents can effortlessly access the laundry status at a glance. Each washer is thoughtfully assigned a unique identification number for easy reference. The core feature of this page is the current status column, where users can quickly ascertain whether a washer is currently in use or available. The visual representation of this data is designed for clarity and ease of understanding. For this purpose, when a washer is in use, the status is prominently displayed in red, making it instantly recognizable. Conversely, when a washer is available, the status is presented in green in a separate row, providing a clear visual indicator.

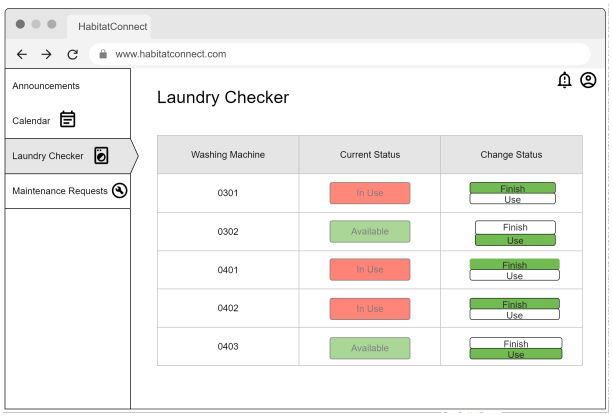


Fig. 27: Laundry checker page (mock up)

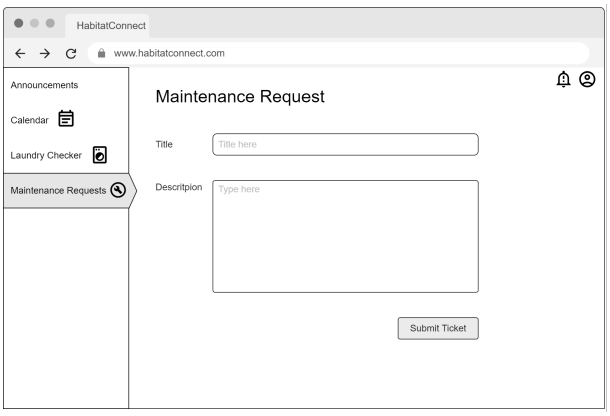


Fig. 28: Submit ticket page (mock up)

As it has been previously mentioned, the laundry checker page involves the following features:

- **Current** status: there is no action to be done in this row. It is simply to show the status of the washer and is completely dependent on the change status row
- **Change** status: users can perform two actions, either click the finish button or the use button. There is no limit on when users can use these buttons. Furthermore, both buttons are free to use and are dependent on the washer's physical status.
  - `Finish`: once clicked, the status of the corresponding washing machine will automatically change to available
  - `Use`: once clicked, the status of the corresponding washing machine will automatically change to in use

J. Maintenance requests

Upon opening the maintenance requests tab on the taskbar for the first time, the user will see the message *You do not have any requests at this moment*, with a create button next to the message. Once the user clicks this button, they will be taken to a new page in which they can perform a maintenance request. The requests made by the users contain the following fields, and are affirmed with input validation:

- `Title`: string of maximum 500 characters
- `Description`: string of maximum 1,000 characters

Once the user clicks the submit request button, a popup will occur, confirming the user's maintenance request in the form of a ticket with the message *Thank you for submitting your ticket. Your request will be closed in the next three days*. Then, the user will be redirected to the maintenance requests page, where they will see a table showcasing the ticket title, description and update tab. On the system's side, the ticket is stored in the MongoDB database with an expiration date of three days after the creation date. Moreover, users will also have the option to update their ticket information (title or description). Multiple requests can be made, and the newest ticket will be at the top of the table. There is no maximum number of tickets a user can open, but once the user has created more than 5 tickets, a page switch button will appear so that the user can switch

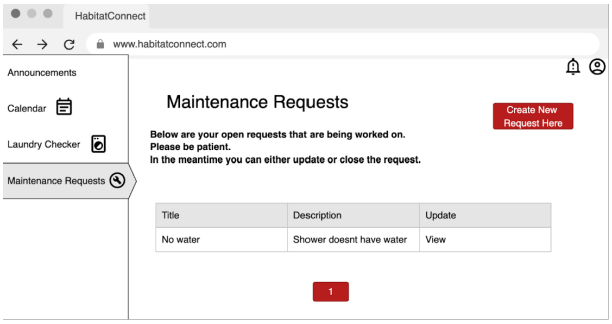


Fig. 29: Example of a submitted ticket (mock up)

between request table pages. The ticket, once submitted, will be automatically sent to an admin/apartment manager in the form of an email. At this point, the maintenance issue should be verbally deliberated between the tenant and admin. Once the maintenance request has been solved, it will be displayed as a completed ticket.

V. Architecture Design & Implementation

A. Overall Architecture

In this subsection, we will provide a detailed description of the overall system architecture. In the figure below, we can see the relationship between the frontend, backend, database and authentication mechanism. The system operates through a three-tier architecture, with a frontend using HTML, CSS, Bootstrap, and EJS communicating bidirectionally with a NodeJS and Express backend. The backend manages interactions with a MongoDB database through Mongoose for data storage and retrieval. Authentication is managed by PassportJS and Google Authentication, securing user access bidirectionally by interfacing with both the backend and the MongoDB database. The frontend triggers HTTP requests and API calls to the backend, initiating data processing, storage, and retrieval. This design allowed us to design a responsive web application with secure user authentication and efficient data flow between the components.

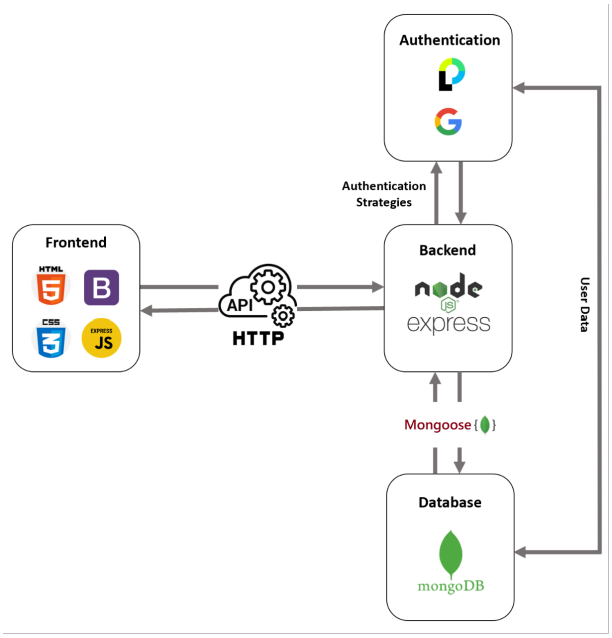


Fig. 30: Architecture schema

B. Directory Organization

By organizing our directory structure, we can maintain a clean and scalable codebase. We have defined a clear separation between frontend and backend components, allowing us to enhance code readability while streamlining the development and collaboration. Our overall directory structure is:

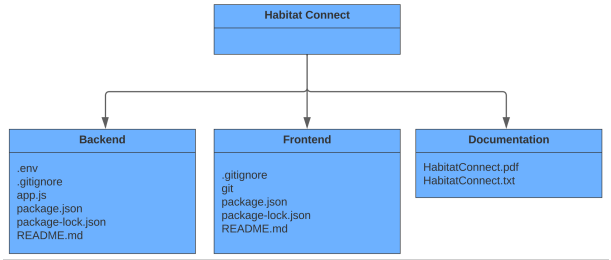


Fig. 31: Overall directory organization

Module 1: Frontend

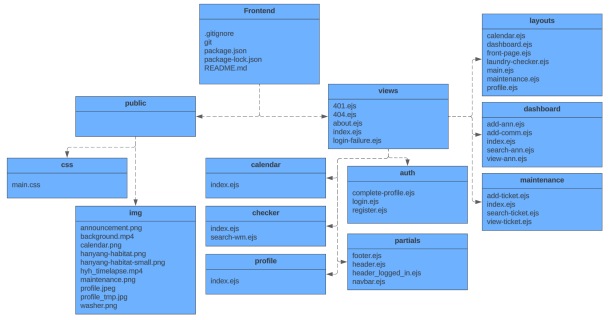


Fig. 32: Frontend module (See Table IV)

1) Purpose: The frontend module is our project's user-facing interface, which is intended to be engaging and intuitive. The purpose of the frontend is enabling possibilities for creating announcements and repair requests, as well as real-time monitoring of washing machine statuses and more. This module ensures a seamless user experience by presenting data fetched from the backend, with client-side validation delivering quick feedback. It connects to backend APIs to do tasks including creating, updating, and deleting announcements and maintenance requests. The interface is responsive and not dependent on any specific devices, with a component-based architecture for modular development and scalability. The use of specific visual components improves the entire user experience by making interactions with announcements, calendar, maintenance requests, washing machine status, and profile setting more dynamic and interesting.
2) Functionality: The frontend functionality is crafted using a combination of HTML, CSS, and JavaScript, providing a solid foundation for the user interface. HTML structures the content, while CSS adds style, ensuring a visually appealing design. JavaScript, along with the EJS templating engine, provides dynamic content updates and interactivity. Bootstrap, a powerful CSS framework, enhances responsiveness and streamlines the design process with pre-built components. The integration of these technologies results in a maintainable structure, enabling efficient development. The application's frontend leverages AJAX for asynchronous data retrieval, ensuring a seamless and responsive user experience. Furthermore, responsive design principles are implemented to guarantee compatibility across various devices, making it easy for users to create announcements, submit maintenance requests, and more functionalities effortlessly.
3) Location of Source Code: Can be found in `/HabitatConnect/Frontend`
4) Class Components:
    a) General pages
        - 401.ejs: this page represents an *Error 401 - Unauthorized* response, indicating that the user lacks the necessary credentials to access a specific resource. It displays an error message and an explanatory paragraph. Users are guided to navigate back to the main page through a back to main button
        - 404.ejs: this page presents an *Error 404 - Not Found* message, indicating that the requested resource is not available. It displays an error message along with a descriptive paragraph. Users are guided to either check the URL or easily navigate back to the homepage using the back to main button
        - about.ejs: this is the about page for our project. It is intended to use it at the moment but it will

be prepared for future purposes. It will contain information about the team and the goal of this project
- index.ejs: this is the landing page of our website. Since it is the first page that the user sees it contains the login and register buttons. The user gets greeted with a welcome message and a timelapse video of Wangsimni is running in the background
- login-failure.ejs: the user gets redirected to this error page in case the login fails. It shows a message with possible causes for the failed login and provides a button to redirect the user back to the login

b) public/css & public/img
- main.css: styling sheet for all the frontend components of this project
- /img: here are all images and videos stored that are being displayed in the user interface

c) views/calendar
- index.ejs: this page represents a calendar. It uses CSS styles to structure and format the calendar layout. The calendar dynamically generates date boxes using EJS and Node.js, highlighting birthdays based on user data. JavaScript functions enable navigation to previous and next months, allowing users to interact with the calendar

d) views/checker
- index.ejs: this page features a table displaying information about each washing machine, including its name, current status, and options to change its status. If there are washing machines available, the table is populated dynamically based on the data received. Users can see the current status of each machine and choose to either mark it as in use or finished using corresponding buttons. These buttons trigger form submissions, updating the status of the selected washing machine. If there are no washing machines available, a message with a small logo apologizes and suggests returning to the dashboard
- search-wm.ejs: this is a search bar component that serves as a search results display within the washer page. If search results are available, it shows a list of clickable washer titles, each linking to the corresponding washer. If there are no search results, it displays a simple message indicating that no items were found.

e) views/profile
- index.ejs: the profile page provides users with the ability to view and update their profile information. The page welcomes the user by their username and includes a form for updating various details. If there are any validation errors, they are displayed at the top in a red alert box. The form includes fields for updating the username, room number, and birthdate. Additionally, users can upload or change their profile picture. If a profile image is already set, it is displayed; otherwise, a default image or placeholder is shown. The update button triggers the form submission for updating the profile

f) views/auth
- complete-profile.ejs: this page is a form for users to submit essential details required to finalize their profile. The form includes input fields for a username, room number, and birthdate. Error handling is implemented to display specific error messages if any issues arise during form submission
- login.ejs: this is the login page where the user can either login with an email/password combination or use the Google sign in feature
- register.ejs: this page is used to create a new user in the system using either an email/password combination or the Google sign in information. Afterwards the user gets forwarded to complete their profile details

g) views/partials
- footer.ejs: this is the footer element containing basic information about this project. The footer can be reused in the layouts for all pages
- header_logged_in.ejs: this is the header element including a title and search-bar. This specific header is only being displayed to logged in users. The difference is that instead of the login button there is a logout button. Reusable in all the layout pages
- header.ejs: this is the basic header element that is being shown to not logged in users, showing the login and register buttons. It is also missing the search-bar that is only displayed to logged in users. Used in the layout pages
- navbar.ejs: this is a vertical navigation bar that provides navigation buttons consisting of title and icon. The navigation bar is being reused in the layout pages as a component like the header and footer for example

h) views/maintenance
- add-ticket.ejs: this page serves as the submission form for maintenance requests, allowing users to submit tickets with a title and description. It includes form validation to ensure that the title and description do not exceed specified character limits (500 characters for the title and 1000 characters for the description). If the form submission is successful, a modal (addModal) appears to confirm the submission, providing information about the expected closure of the

request within the next three days. In case of an error, such as empty fields or exceeding character limits, an error modal (errorFormModal) is displayed, guiding users to correct the form issues
- index.ejs: this page displays maintenance requests, providing users with information about their open requests that are currently being worked on. Users can create new maintenance requests using the create button if they don't have any open requests. For users with open requests, the page shows a table with the title and a truncated description of each request, along with an option to view/update each request. The table includes pagination for easy navigation through multiple requests. If a user doesn't have any open requests, a message encourages them to create a new request using the provided button
- search-ticket.ejs: this is a search-bar-component that serves as a search results display within the maintenance requests. If search results are available, it shows a list of clickable titles, each linking to the detailed view of the corresponding requests. If there are no search results, it displays a simple message indicating that no items were found
- view-ticket.ejs: this page allows users to view and update a specific maintenance request. The navigation provides a link back to the maintenance request page and displays the title of the current ticket. The page includes a form for updating the ticket's title and description. Users can submit changes using the update button, and there's an option to close the ticket using the close button, which triggers a confirmation modal. If there's an error, it will be displayed at the top of the page in a red alert box

i) views/layouts

- calendar.ejs, dashboard.ejs, front-page.ejs, laundry-checker.ejs, main.ejs, maintenance.ejs, profile.ejs: the layout structure of this project is designed for modularity and consistency across various views. Each layout follows an HTML structure, incorporating EJS for dynamic content rendering. The ¡head¿ section includes dynamic metadata, such as the page title and description. External styles are applied through Bootstrap v5.3.2 and a project-specific stylesheet. Google Fonts provides more design-choices. The layout is organized into sections: header, navbar (or sidebar), main content area, and footer, allowing for the inclusion of dynamic content through EJS partials. This approach promotes code reusability and a unified user interface with the ability to adjust each page's layout easily

j) views/dashboard

- add-ann.ejs: this page displays a form to add a new announcement to the announcement board. The form includes a title and textbody that gets sent to the backend and validated
- add-comm.ejs : users are able to post comments under a posted announcement. This page includes a form with a textbox that sends the comment to the backend and gets validated
- index.ejs: this page is a dynamic dashboard displaying announcements. It starts with a greeting to the user and offers a create button to add new announcements. The main content area shows a grid of announcements, each represented by a bootstrap card with a title and a preview of the content. If there are no announcements, it prompts the user to create their first announcement with a call-to-action link. The page includes a pagination feature at the bottom to navigate through the announcements efficiently
- search-ann.ejs: this is a search bar component that serves as a search results display within the dashboard. If search results are available, it shows a list of clickable titles, each linking to the detailed view of the corresponding announcement. If there are no search results, it displays a simple message indicating that no items were found
- view-ann.ejs: this page serves as the view for a specific announcement within the dashboard. It displays the announcement's title, body, and creator, and provides functionality for the announcement owner to update or delete the announcement. Users who are not the announcement owner can only view the announcement details. Additionally, the page includes a comments section, allowing users to view existing comments and, if they are the comment creator, update or delete their comments

5) HOW AND WHY WE USE IT? HTML, CSS, Bootstrap, JavaScript, and EJS form a robust frontend toolkit. HTML structures content, CSS styles layouts, and Bootstrap ensures consistency. JavaScript adds interactivity, enabling dynamic features. EJS, integrated with Node.js, dynamically generates HTML based on server-side data. Since we have Android and iOS users in our group we decided against developing a mobile application and instead chose to focus on a web-application that can provide us with new programming experiences and help us in future projects.

MODULE 2: BACKEND

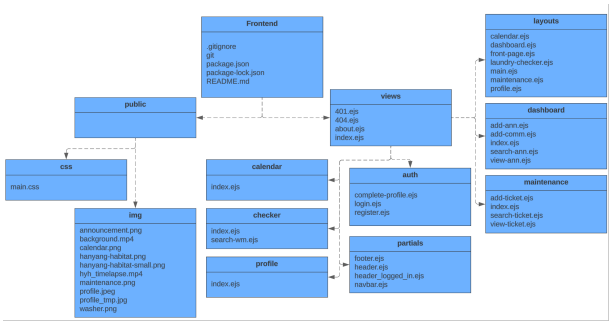


Fig. 33: Backend module (See Table V)

1) PURPOSE: The backend module forms the backbone of our project, focusing on server-side operations to handle requests from the frontend. It plays a pivotal role in managing and maintaining data integrity for user profiles, announcements, maintenance requests, and washing machine statuses. Through robust APIs, it facilitates seamless communication with the database, ensuring secure and scalable functionality. This backend system is designed to support critical processes such as user authentication, data storage, and retrieval, laying the foundation for a dynamic and engaging user experience crafted by the frontend module.
2) FUNCTIONALITY: The backend module's functionality is made possible using Node.js, Express.js and Mongoose to handle server-side logic and data management. Its primary purpose is to process requests from the frontend, providing seamless communication with the database. Node.js ensures an architecture for efficient handling of concurrent operations. Express.js simplifies route handling and streamlining the development of APIs. Mongoose, a MongoDB object modeling tool, provides an elegant interface for interacting with our database, enhancing data integrity and retrieval. This backend system supports user authentication, manages data persistence for user profiles, announcements, maintenance requests, and washing machine statuses. By using these technologies, we achieve a scalable, efficient, and secure backend that complements the frontend's dynamic user experience.
3) LOCATION OF SOURCE CODE: Can be found in `/HabitatConnect/Backend`
4) CLASS COMPONENTS:
   a) Environment settings
      - .env: this configuration file holds key-value pairs crucial for our application's functionality. It stores sensitive information such as the MongoDB URI, which connects our backend to the database hosted on MongoDB Atlas. The Google Client ID and Secret are essential for enabling Google OAuth, ensuring secure user authentication. The specified callback URLs, both for registration and login, are important for handling the redirection and authentication processes when users sign up or log in using their Google accounts
      - app.js: the app.js file serves as the entry point for our Node.js Express application, managing various components. It employs middleware like express-ejs-layouts, method-override, and express-flash to enhance routing and manage HTTP methods effectively. The configuration includes sessions, utilizing express-session with a MongoDB store for persistent user data. Passport.js is integrated for authentication, enabling secure user management. Routes are organized and directed to respective controllers. The templating engine EJS is employed to structure dynamic views. Connection to the MongoDB database is established through the connectDB function. The file is structured to handle various routes, such as authentication, index, calendar, dashboard, laundry checker, maintenance, and profile. It also ensures a 404 response for any unrecognized paths. The application listens on the specified port
   b) server/config
      - db.js: this config file uses Mongoose to connect a Node.js application to a MongoDB database specified by the `MONGODB_URI` environment variable. This module is crucial for maintaining a connection to the database, ensuring seamless data operations within the backend
   c) server/middleware
      - checkAuth.js: this middleware component checks whether the user is authenticated. If the user is authenticated, it allows the request to proceed to the next route handler. If the user is not authenticated, it renders a 401 Unauthorized view, indicating that the user lacks the necessary credentials to access the requested resource
   d) server/routes
      - auth.js, calendar.js, dashboard.js, index.js, laundryChecker.js, maintenance.js, profile.js: all these routes are part of the system, providing the necessary endpoints like for example: user registration, login, dashboard access, maintenance ticket creation and many more. The actual implementation details are in the respective Controller module, which contains the logic for each route
   e) server/controllers
      - authController.js: this controller handles user registration, login, and profile completion. It supports both local and Google-based registration and login. For local registration, users provide an email and password, which is securely hashed and stored in the database. Google

registration and login use OAuth2.0, allowing users to sign in with their Google accounts. The system also ensures a complete user profile with validation for username, room number, and birthdate. Users can update their profiles, and errors are displayed when validation fails

- calendarController.js: this controller provides user birthdays and the current date. User data, including user IDs, usernames, and birthdates, is fetched from the database
- checkerController.js: this controller manages the logic related to washing machines, such as retrieving their status, updating status based on user input, and handling search functionality. Additionally it controls the amount of washing machines that should be shown per page
- dashboardController.js: this controller, associated with announcements and comments in the dashboard page, handles views for displaying, updating, and deleting announcements. Users can add comments, update them and perform searches. The controller ensures length validation for titles, bodies, and comments, redirecting appropriately after each operation
- mainController.js: this controller manages views for the homepage of the web application. The homepage displays the main content with a layout for the front page
- maintenanceController.js: this controller manages maintenance-related functionalities, including viewing, updating, deleting, adding and searching tickets. The maintenance view displays a paginated list of maintenance tickets for the user, with options to view, update, and delete each ticket. Users can also add new maintenance tickets, search for specific tickets, and view detailed information about a specific ticket. The controller ensures that users can only access and modify their own tickets
- profileController.js: this controller handles profile-related functionalities. The profile view displays the user's profile information, including username, room number and birthdate. Users can update their profiles by submitting the provided form, and the controller validates the input data, ensuring that the username is within a certain character limit, the room number is within a specified range and the birthdate is in a valid format and not in the future. If there are errors in the input data, the controller renders the profile view again with error messages. If the update is successful, the user is redirected to the profile page

f) server/models

All the models are designed to interact with the appropriate MongoDB collection in a Node.js application using Mongoose.

- Announcements.js: this model represents announcements in a MongoDB database. It includes fields for the announcement's owner, title, body, creation date, last update date, and an array of references to associated comments

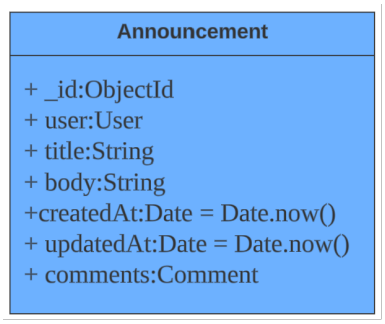


Fig. 34: Announcement schema

- Comment.js: this model represents announcement comments. It includes fields for the ticket's owner, title, body, creation date, and last update date

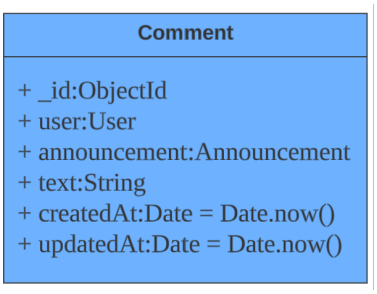


Fig. 35: Comment schema

- Ticket.js: this model represents maintenance tickets. It includes fields for the ticket's owner, title, body, creation date, and last update date. Additionally, the model is configured with an index to automatically expire tickets after 3 days, enhancing data management

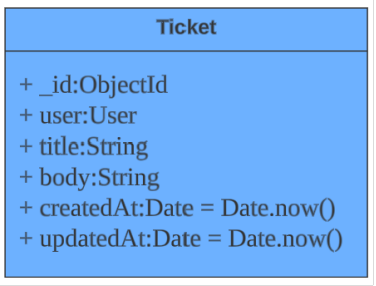


Fig. 36: Ticket schema

- User.js: this model includes fields such as email, password, googleId, username, profileImage, birthdate, and roomNumber. Each field has specific validation criteria. The model is designed to manage user information, providing a structured schema for registration, authentication, and profile management within the application

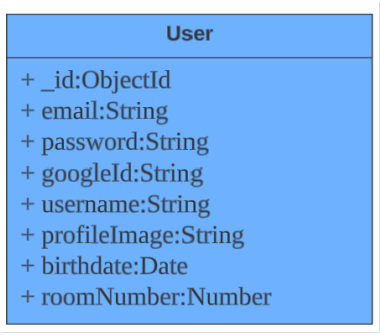


Fig. 37: User schema

- WashingMachine.js: this model represents the status of a washing machine. It includes fields for status and updatedAt. The status field has possible values of available and in use, with a default value of available. This model is designed to manage and track the status of washing machines in the application

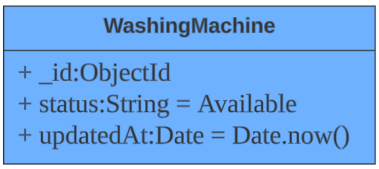


Fig. 38: Washing Machine schema

5) How and why we use it? We chose this backend technology because it aligns with modern web development practices, emphasizing scalability and maintainability. In Visual Studio Code, we use Node.js and Express for development. This helps us write and organize code easily. Opting for Node.js and MongoDB helps us gain valuable experience with these relevant technologies for future projects.

Module 3: Database - MongoDB

1) Purpose: MongoDB serves as the primary database solution for our application, providing a flexible and scalable storage system for managing various data types.
2) Functionality: MongoDB is a NoSQL database that excels in handling unstructured and semi-structured data. It supports a document-oriented data model, allowing us to store and retrieve complex data structures with ease.
3) Location of Source Code: Can be found in `/HabitatConnect/Backend`
4) Class Components: The components consist of schemas and models that define the structure and behavior of the data stored in MongoDB. These components include schemas for users, announcements, comments, tickets, and washing machines like mentioned in the backend part.
5) Where is it taken from? MongoDB is an open-source, document-oriented database system. We utilize the official MongoDB-Node.js driver and Mongoose, an ODM (Object Data Modeling) library for MongoDB and Node.js, to interact with the database.
6) How and why we use it? We chose MongoDB for our database because it's flexible and can handle a lot of different types of data. This is important for us because our data needs might change over time. MongoDB's way of storing data is similar to how JavaScript works, which makes it easy to connect with our Node.js backend. To manage documents directly we use the cloud based MongoDB Atlas where everything can be configured. Using MongoDB fits well with our goal of creating a modern and scalable web app, making sure users have a smooth and responsive experience.

VI. Use Cases

In this section, we will provide a detailed description of some use cases that demonstrate all requirements implemented. Some examples of questions we aim to answer are: how does a user register in Habitat Connect? What happens if the validation check is not successful? How can I user comment on an announcement? We will explore these and more practical scenarios in detail to get a better understanding of our software behavior.

A. Registration and Log In

1) Log In: if the user clicks the login button in the top right corner, the login form will be shown. The user can either log in with an email/password combination or choose the sign in with Google option via the button below.

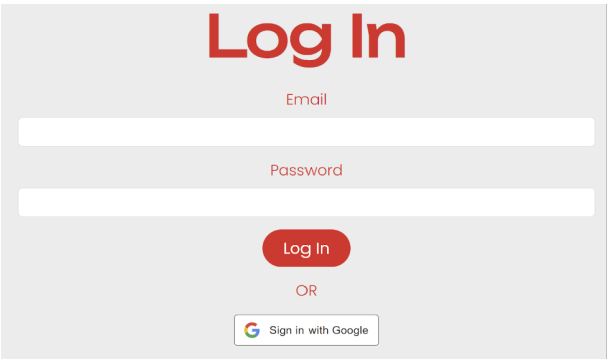


Fig. 39: Login page

If the user decides to login using Google OAuth, they will need to choose an existing Google account before accessing Habitat Connect's dashboard.

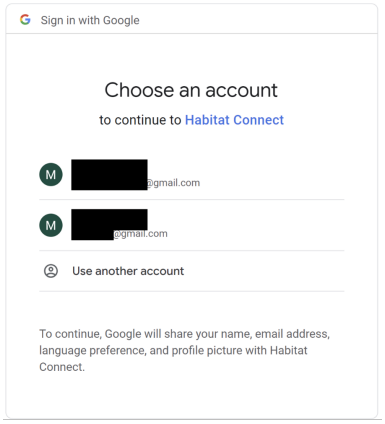


Fig. 40: Login via Google OAuth

On the other hand, if the user would like login locally, they will need to provide their email and password. If the form is not filled out correctly (wrong email format or empty textboxes) the user receives visual feedback of what is missing or wrong after clicking the login button. Moreover, the user will not be able to complete the login process until the form is successfully stored.

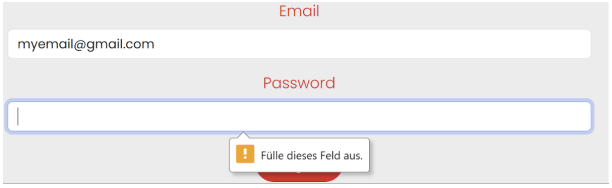


Fig. 41: Empty login form

If the given email does not exist in the database or the password processed does not match the one found in the database, the user will see the following error page.

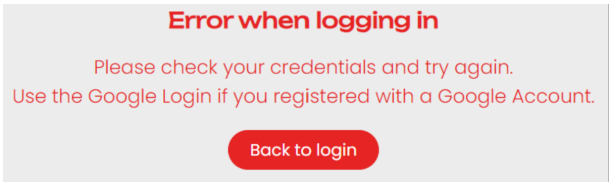


Fig. 42: Login failure example

Once the user is successfully logged in, they get redirected to the announcement board.

2) Register (Connect now!): if the user clicks the register or *Connect now!* button, the register from will be shown. It looks and works the same as the login page with the validation.

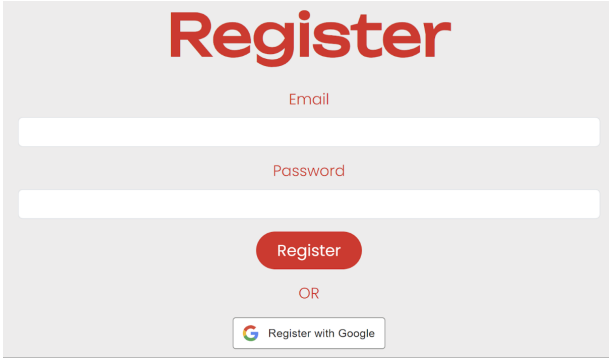


Fig. 43: registration page

If the email already is registered or something went wrong while processing, the following message gets shown and the user can try again.

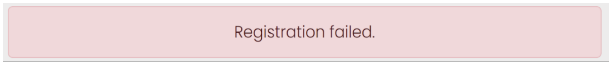


Fig. 44: Registration failure error

3) Log Out: the user has the ability to log out of their account by pressing the logout button that is visible on every page in the top right corner. Once clicked, they will be redirected to the homepage.

B. *Logged In Restriction*

If the user is not logged in and tries to access Habitat Connect's protected pages, they will be redirected to the unauthorized page. The only action available for the user is the button to be redirected to the homepage.

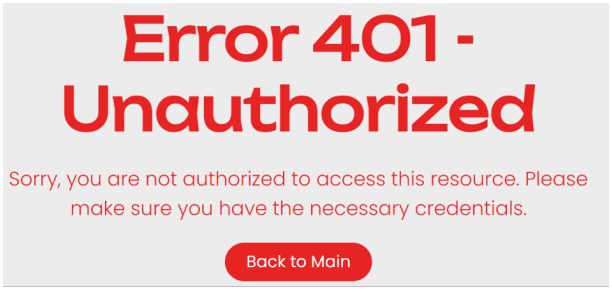


Fig. 45: 401 - Unauthorized page

C. *Profile*

1) Complete Profile: if the registration with the mail/password or google authentication was successful, the user is redirected to the complete profile page. This page is necessary to complete the profile of the account created in the database.

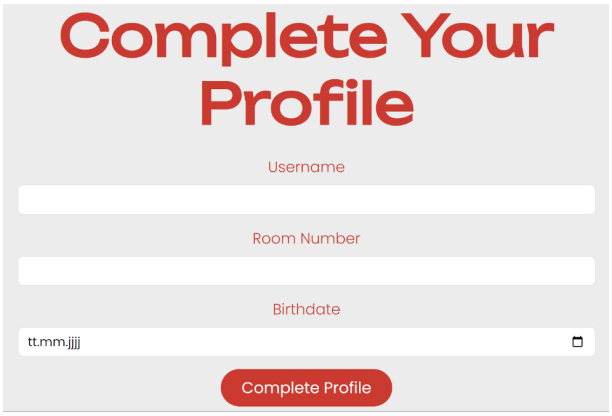


Fig. 46: Complete profile page

A text box is used for the username where the character limit is 12, if it gets exceeded then the following message will be shown to the user after trying to complete the profile via the complete profile button.

Fig. 47: Username validation error

The room number is a number where the user can press the up and down arrows to select a number or just type it in but it has to be within 300 to 500 or else the error message below pops up when trying to complete the profile via the complete profile button.

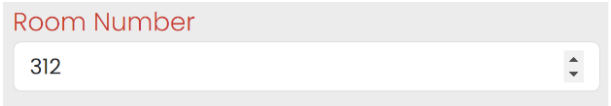


Fig. 48: Room number cell

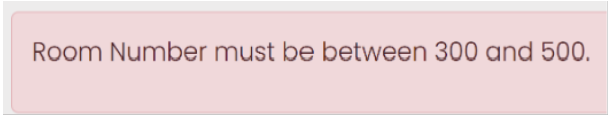


Fig. 49: Room number validation error

To change the birthdate the user can either type it in themselves or use the calendar icon and select the appropriate date.

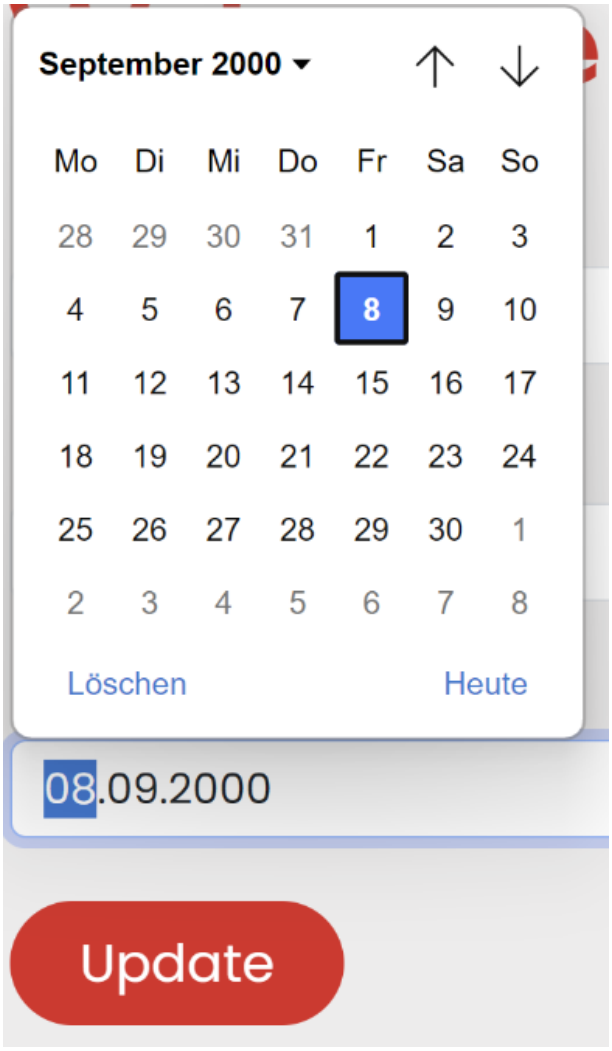


Fig. 50: Birthdate cell

If the user selects a date in the future and tries to complete the profile the following message will be shown.

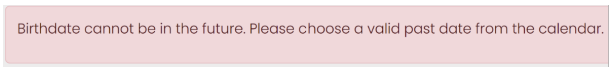


Fig. 51: Birthdate validation error

When the user fills out the fields correctly and presses the complete profile button, the user gets forwarded to the announcement board.

2) Profile View: to access the profile view, the user presses the profile tag in the navigation bar. Once pressed, the user gets forwarded to the profile overview where the username, room number, birthdate and the profile picture is displayed.

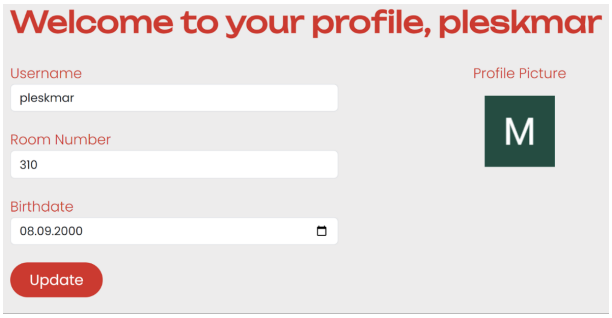


Fig. 52: Profile page

3) Update Profile: the user is able to update the username, room number and birthdate if needed. Here the same validation messages will be shown as when creating the profile. To change any of the fields, the user clicks on the desired field and types in the new value. If the user changes the wanted fields correctly and presses the update button the changes will be saved and directly updated on the page.

D. Dashboard

The first page the user gets to see is the announcement page where 8 announcements are displayed per page. The announcement page features pagination where the user can click either the page index to access a certain page directly. In the use case found below, users can click on the 2 button to access the second page of the dashboard, or click on the next to button cycle through to the next pages. The same logic applies when accessing the previous page with the previous button.

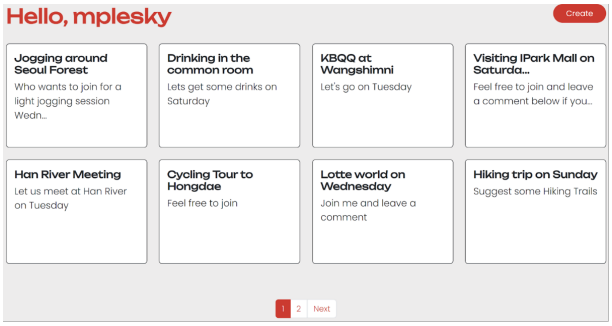


Fig. 53: Dashboard

If there are no announcements yet, a message will be shown to the user that the first announcement can be created with clicking on the create button.

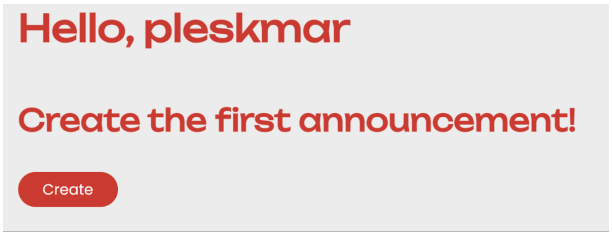


Fig. 54: Empty dashboard page

1) **Post** an announcement: To create an announcement, the user presses the create button in the top right corner of the announcement page and gets forwarded to the add announcement page. If the user decides not to post an announcement, there is the possibility to click the on dashboard link above the form that will lead the user back to the dashboard.

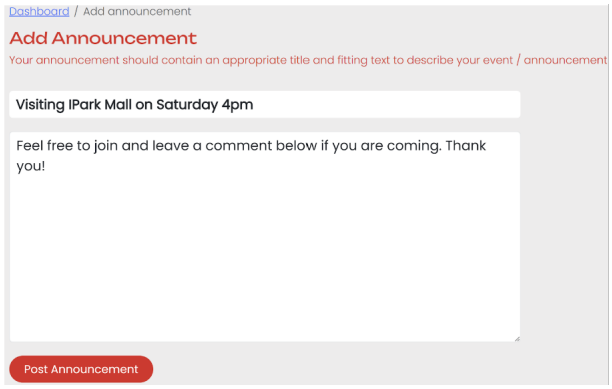
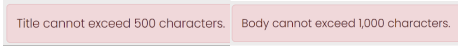


Fig. 55: Add announcement page

Here the user has to fill out the title (max 500 characters) and description (max 1000 characters) fields. If these requirements are not met when pressing the post announcement button the corresponding error messages pop up.



(a) Title validation error (b) Body validation error

Fig. 56: Validation errors on add announcement page

If the form is empty or one field is missing and the user tries to post the announcement the user gets informed directly from the form validation that something is missing. Once everything is valid and the user clicks the post announcement button, the user gets forwarded to the announcement page and the new announcement is visible on the first position, since it is the most recent one as seen below.

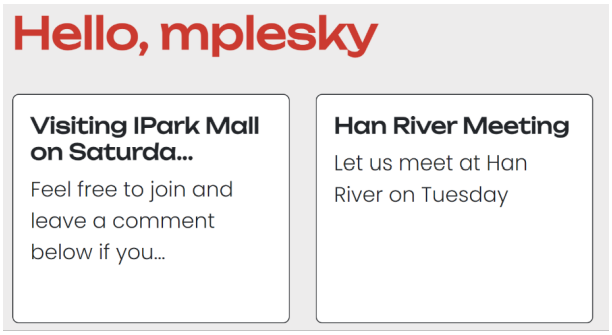


Fig. 57: Example of posted announcement

2) **View** an announcement: all announcements are displayed in rectangular form that serve as buttons. If the user clicks on one of the rectangles, the user gets redirected to the announcement view. Every user can see all announcements posted.

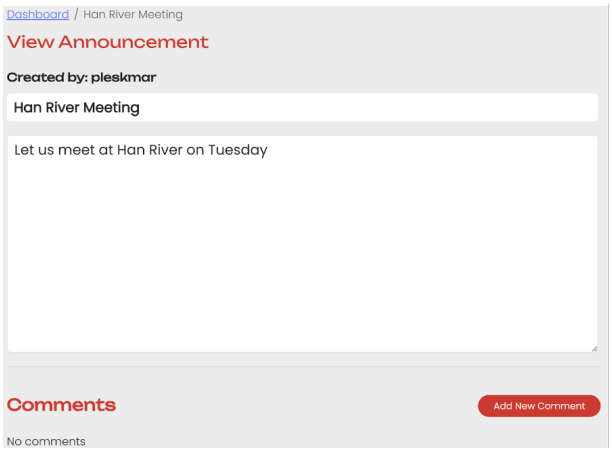


Fig. 58: View announcement page

3) **Update** an announcement: the creator of the announcement has the ability to update the title and text of their announcement. Moreover, the same validation criteria applies as when deleting an announcement.

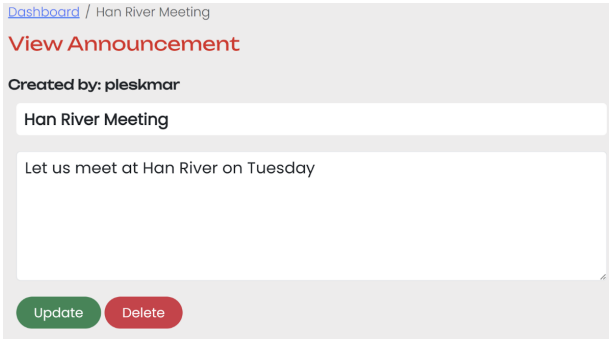


Fig. 59: Update announcement example

When a user updates an announcement, they will be redirected to the dashboard where they will see their updated announcement at the top of the dashboard. However, if the title and body lengths of the updated announcement are not valid (500 and 1,000 characters respectively), the owner will see some validation errors on the screen before they are able to successfully update the announcement.

4) **Delete** an announcement: the creator has the ability to delete his announcement with the delete button. Once the delete button is pressed a confirmation popup will show.

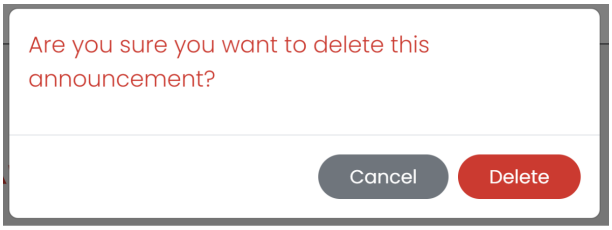


Fig. 60: Delete announcement popup

If the creator does not want to delete the announcement, the cancel button can be pressed. To delete the announcement, the creator needs to confirm by clicking the delete button. Once deleted, the creator gets forwarded to the announcement page and the announcement will not be shown anymore since it will be erased from our database.

5) **Comment** on an announcement: each announcement can have comments created by users. If the user clicks on an announcement that has no comments yet, the following message will be displayed.

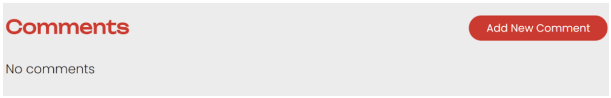


Fig. 61: No comments on an announcement

To add a comment the user presses the add new comment button which leads them to the comment creation page.

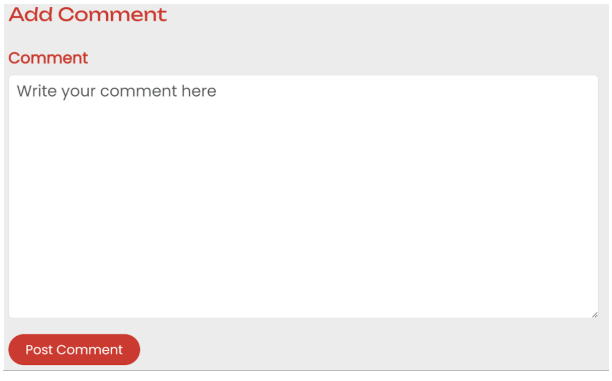


Fig. 62: Add comment page

The comment can not be empty or exceed 500 characters.

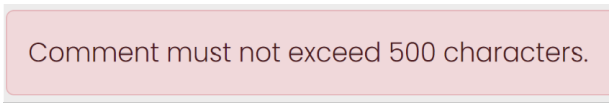


Fig. 63: Comment validation error

When the comment entered by the user passes the validation check and the post comment button is pressed, the user gets forwarded to the announcement commented on. The comment is now visible.

Fig. 64: Comment example when viewer is owner

As the creator of the comment, there is the possibility to update or delete the comment. Once the comment is adjusted correctly, the creator can click the update comment button to save the changes. If the comment exceeds 500 characters, the appropriate message will be displayed. To delete the comment, the creator can press the delete comment button. Once deleted, the comment will no longer be displayed since it will be removed from the database. If, on the other hand, the viewer is not the creator, the comment will be displayed without the update textbox and update/delete buttons.

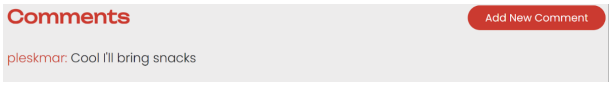


Fig. 65: Comment example when viewer is not owner

6) **Search** an announcement: users have the ability to search for an announcement by inserting some keywords in the search bar found in the dashboard, and view announcement pages. They can either search for keywords in the title or body of an announcement. For example, if the user inserts the keyword *Meeting* in the search bar, they will see all announcements that contain the inserted keyword. Furthermore, users have the possibility to click on the link to be redirected to the view announcement page, where they will be able to see the complete announcement, as well as any comments posted.

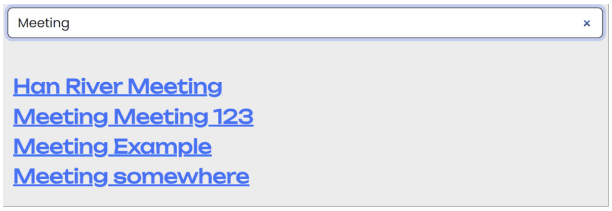


Fig. 66: Announcement keyword search example

If the user searches for something that does not exist like *nothing*, the *No items found* message will be shown.

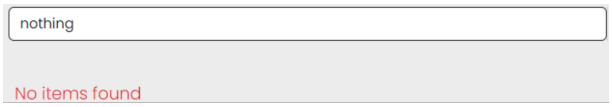


Fig. 67: Search with no items

E. Calendar View

To access the calendar view, the user presses the calendar tag in the navigation bar. Once pressed, the calendar page shows the current month in the style of a calendar. In the calendar, all birthdays of the registered users are visible. The user has the option to switch back and forth between months to see upcoming or past birthdays. To access the next month, the user clicks the next month button. To access the previous month, the user clicks the previous month button.

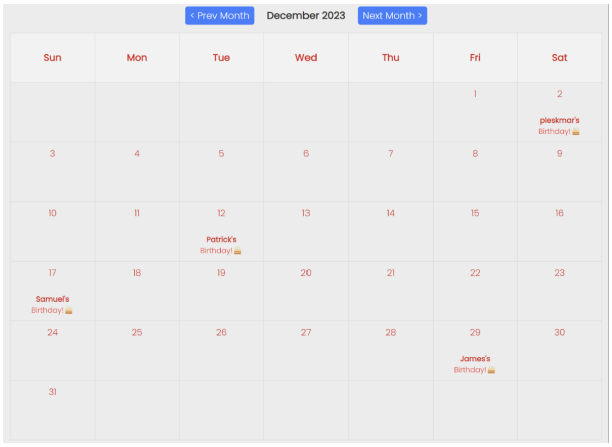


Fig. 68: Calendar view page

F. Laundry Checker

To access the laundry checker view, the user presses the laundry checker tag in the navigation bar. Once pressed the user sees the current status of the washing machines. From here the user can change the status of the washing machines accordingly. When the user starts to use a washing machine the status needs to get updated. To set the washing machine status to in use, the user clicks the red use button. This updates the current status of the washing machine in use. To set the status to available, the user presses the green finish button once the laundry is finished and taken out. This showcases to other users that the washing machine can be used again.

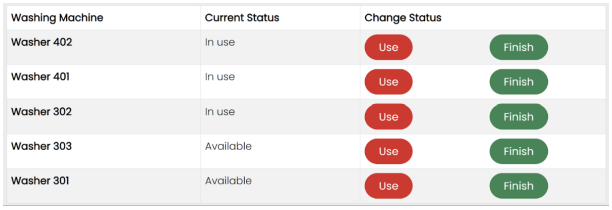


Fig. 69: Laundry checker page

Moreover, the washing machines also can be searched through, where the id and the status is the data given. Users

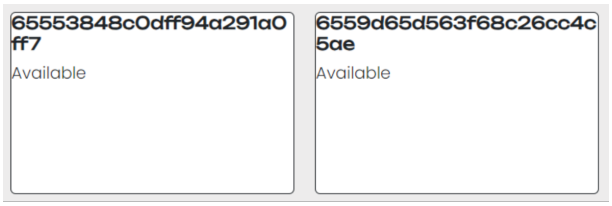


Fig. 70: Search a washing machine

have the option to search a washing machine by its id or status. It is a very useful tool if the user is only interested in available washing machines. So, if the user searches for all available washing machines with the keyword *Available*, it displays washing machines containing the status available. If there are no washing machines containing the search keyword, the user will see *No items found*.

G. Maintenance Requests

To access the maintenance request view the user presses the maintenance requests tag in the navigation bar. If the user has open requests they will be displayed in a table structure.

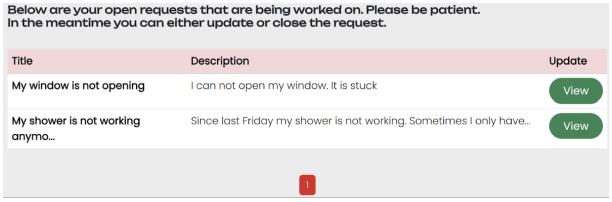


Fig. 71: Maintenance requests page

Else, the user sees the message *You do not have any open requests at the moment* and can create the first request with the create button.

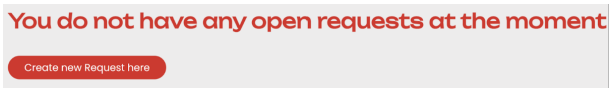


Fig. 72: Empty maintenance requests page

When the user has more than five requests open at the same time, the same pagination feature for the announcements can be used. To navigate the user can press the next button to get to the next set of requests or access directly the second page with pressing on 2 in this case. Once on another page the user can press the previous button to reach the previous set of requests.

1) **Create** a ticket: if the user presses the create button, the following form below will be shown. Here the user has to fill out the title (max 500 characters) and description (max 1,000 characters).

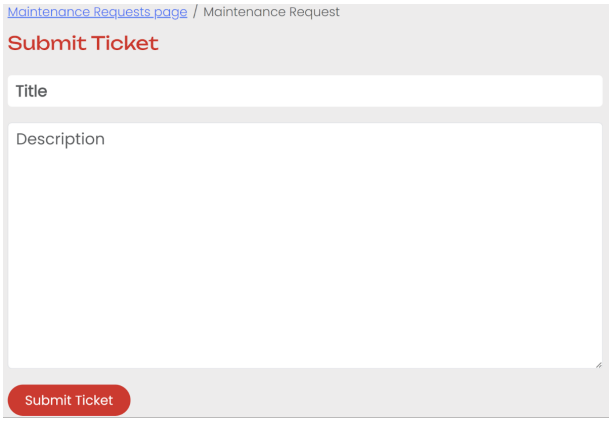


Fig. 73: Create ticket page

If these requirements are not met or the title/description is empty the following error message pops up.

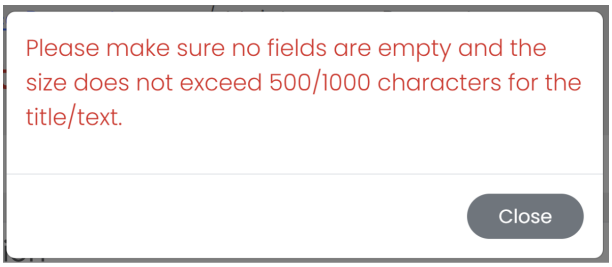


Fig. 74: Ticket validation popup

To close this pop-up the user clicks the close button and the user sees the form again. Once the user gives an appropriate title and description the request can be submitted with the submit ticket button. After submitting a correct request the following pop-up is shown that also can be closed with the understood button.

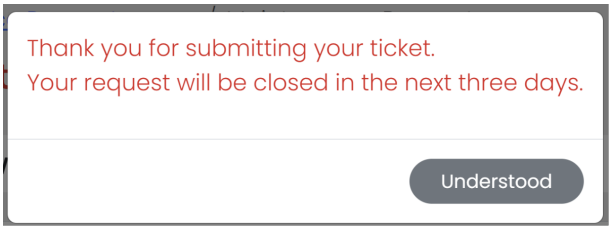


Fig. 75: Submit ticket popup

Once closed the user gets forwarded to the maintenance request overview page and sees the newly created request.

2) **View** a ticket: to view a specific request the user presses the green view button displayed in the last column of the specific request. This takes the user to the ticket where it can be updated or closed manually.

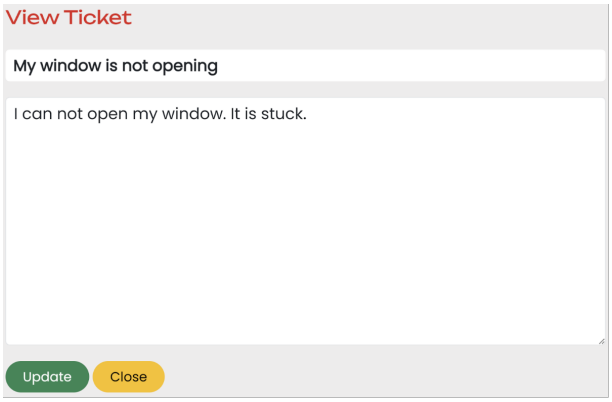


Fig. 76: View ticket page

3) **Update** a ticket: the user can adjust the text where the same validation criteria is valid as when creating a request. After adjusting the title or text the user clicks the green update button to save the changes. If the text exceeds more than 1,000 characters the validation error message gets shown and the user can try again. The same goes for the title but with 500 characters. Once the input form is valid, the user lands on the maintenance requests page and sees the updated request at the top of the table.
4) **Close** a ticket: the user can close the request themselves in case the issue has disappeared or been fixed in the meantime. To close the request the user can click the yellow close button from the view ticket page. This will lead to the confirmation popup with two options given to the user.

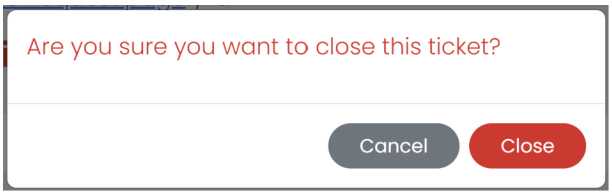


Fig. 77: Close ticket popup

If the user does not want to close the ticket, the cancel button will be pressed. To close the ticket, the user needs to confirm by clicking the close button. Once closed the user gets forwarded to the maintenance request page and the ticket will not be shown in the table anymore since the instance will be deleted from the database.
Recall that when users created a ticket, they could see a pop up message notifying them that the ticket will be automatically deleted from the database three days after the creation date. Therefore, if a user doesn't manually close the ticket, the system will close it automatically.
5) **Search** a ticket: similarly to announcements and washing machines, the user can search for keywords in a ticket, where the title and body are searched through. If the user has an open request containing the word window and searches *Window* in the search bar, the link to the request will be displayed.

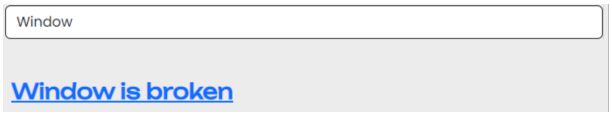


Fig. 78: Search a ticket

If there are no tickets containing the search keyword, the user will see *No items found*.

AUXILIARY TABLES

Directory	File Name
/HabitatConnect/Frontend/	.gitignore git package.json package-lock.json README.md
/HabitatConnect/Frontend/public/css	main.css
/HabitatConnect/Frontend/public/img	announcement.png calendar.png hanyang-habitat.png hanyang-habitat-small.png hyh_timelapse.mp4 maintenance.png profile.jpeg profile_tmp.jpg washer.png
/HabitatConnect/Frontend/views	401.ejs 404.ejs about.ejs index.ejs
/HabitatConnect/Frontend/views/auth	complete-profile.ejs login.ejs register.ejs
/HabitatConnect/Frontend/views/calendar	index.ejs
/HabitatConnect/Frontend/views/checker	index.ejs search-wm.ejs
/HabitatConnect/Frontend/views/dashboard	add-ann.ejs add-comm.ejs index.ejs search-ann.ejs view-ann.ejs
/HabitatConnect/Frontend/views/layouts	calendar.ejs dashboard.ejs front-page.ejs laundry-checker.ejs main.ejs maintenance.ejs profile.ejs
/HabitatConnect/Frontend/views/maitenance	add-ticket.ejs index.ejs search-ticket.ejs view-ticket.ejs
/HabitatConnect/Frontend/views/partials	footer.ejs header.ejs header_logged_in.ejs navbar.ejs
/HabitatConnect/Frontend/views/profile	index.ejs

TABLE IV: Frontend Module Table

Directory	File Name
/HabitatConnect/Backend/	.env .gitignore app.js package.json package-lock.json README.md
/HabitatConnect/Backend/node_modules	(Node.js Project Dependencies)
/HabitatConnect/Backend/server/config	db.js
/HabitatConnect/Backend/server/controllers	authController.js calendarController.js checkerController.js dashboardController.js mainController.js maintenanceController.js profileController.js
/HabitatConnect/Backend/server/middleware	checkAuth.js
/HabitatConnect/Backend/server/models	Announcements.js Comment.js Ticket.js User.js WashingMachine.js
/HabitatConnect/Backend/server/routes	auth.js calendar.js dashboard.js index.js laundryChecker.js maintenance.js profile.js

TABLE V: Backend Module Table

REFERENCES

[1] Facebook for Developers, "WhatsApp Business API - Documentation," Facebook for Developers. [Online]. Available: https://developers.facebook.com/docs/whatsapp/

[2] "KakaoTalk - Services," Kakao Corporation. [Online]. Available: https://www.kakaocorp.com/page/service/service/KakaoTalk?lang=en

[3] "GitHub - Discord," GitHub. [Online]. Available: https://github.com/discord

[4] "Google Calendar API Documentation," Google Developers. [Online]. Available: https://developers.google.com/calendar

[5] "CI Hackathon July 2020 - GitHub Repository," GitHub. [Online]. Available: https://github.com/maliahavlicek/ci_hackathon_july_2020

[6] "Zendesk - GitHub Repository," GitHub. [Online]. Available: https://github.com/zendesk

[7] "Node.js - GitHub Repository," GitHub. [Online]. Available: https://github.com/nodejs

[8] "Node.js - Documentation," Node.js. [Online]. Available: https://nodejs.org/en/docs

[9] "Express.js Documentation," Express.js. [Online]. Available: https://expressjs.com/

[10] "Passport.js Documentation," Passport.js. [Online]. Available: https://www.passportjs.org/docs/

[11] "Bootstrap - GitHub Repository," GitHub. [Online]. Available: https://github.com/twbs/bootstrap

[12] "MongoDB," MongoDB. [Online]. Available: https://www.mongodb.com

[13] "MongoDB - Documentation," MongoDB. [Online]. Available: https://www.mongodb.com/docs/manual/

[14] "Visual Studio Code - GitHub Repository," GitHub. [Online]. Available: https://github.com/microsoft/vscode

[15] "Git," Official Git Website. [Online]. Available: https://git-scm.com/

[16] "Google OAuth 2.0 Documentation," Google Developers. [Online]. Available: https://developers.google.com/identity/protocols/oauth2/web-server

[17] "Overleaf - Documentation," Overleaf. [Online]. Available: https://www.overleaf.com/learn

[18] "Moqups - Online Moqups, Wireframes & Design Tool," Moqups. [Online]. Available: https://moqups.com/

[19] "JavaScript Documentation," Mozilla Developer Network (MDN). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[20] "HTML Documentation," Mozilla Developer Network (MDN). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML

[21] "CSS Documentation," Mozilla Developer Network (MDN). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS]