Enhancing University Study Spaces with a Smart Room Booking Application

Basar CAROVAC
School of Innovation, Design and, Engineering
Mälardalens University, Västerås, Sweden

Benjamin MARGUIN
School of Innovation, Design and, Engineering
Mälardalens University, Västerås, Sweden

Danjel FRROKAJ
School of Innovation, Design and, Engineering
Mälardalens University, Västerås, Sweden

Felix ÅHLÉN
School of Innovation, Design and, Engineering
Mälardalens University, Västerås, Sweden

Ivan BLAZANOVIC
School of Innovation, Design and, Engineering
Mälardalens University, Västerås, Sweden

Momir SAVIC
School of Innovation, Design and, Engineering
Mälardalens University, Västerås, Sweden

Abstract

This project report presents the design and development of BookIT, a smart room booking system tailored for Mälardalen University in Västerås and Eskilstuna, Sweden. The system addresses limitations in existing solutions such as KronoX, which lack intuitive interfaces, smart allocation, and community-driven features. Our approach integrates real-time availability updates, fair allocation algorithms, and event-based booking to promote balanced room usage and improve user experience.

The application was developed using Kotlin with Jetpack Compose for the frontend, ensuring a responsive and modern UI. Core functionalities include room search with filters, calendar-based booking, reservation management, and a community board for event creation and participation. Mapbox integration provides an interactive campus map across multiple floors, enabling users to easily locate available rooms. Authentication and role-based access were implemented through JWT and Passport.js, while Express.js with Prisma handled backend logic, database interactions, and security features such as encrypted credentials and HTTPS enforcement.

Collaboration tools including Figma, Jira, Git, and Zoom facilitated distributed teamwork, ensuring consistency in design and efficient task allocation. Feedback during supervision meetings highlighted improvements in search intuitiveness, event management, and overall usability, which were incorporated into iterative development. The results demonstrate a scalable, secure, and user-friendly booking platform with strong potential for expansion and adaptation to other academic institutions. Future work will focus on refining allocation algorithms, addressing identified security issues, and enhancing the user experience through extended personalization and testing.

Keywords: *smart booking system; resource allocation; real-time availability; university spaces; user experience; mobile application.*

Introduction

BookIT, an application for locating and reserving available workspaces, incorporates real-time availability updates and fair allocation algorithms to suggest rarely used rooms. Providing a calendar feature for an overview of reservations and the available slots. Push notification system to send reminders about reserved spaces. Detailed campus map to display available rooms, a total of three floors. Exclusively utilizes the Västerås and Eskilstuna campuses of MDU, with the possibility of easily interchanging for different buildings.

The initial group meeting was directed towards better understanding the project's requirements and design of the app. The communication was carried out using Zoom meetings online and several in-person meetings located in the university's library. All group members were always present, providing discussion topics as well as solution brainstorming. The plan and design phase was carried out before the first supervision presentation. The design was created using Figma, providing a clear roadmap for the frontend development so that our group members adhere to a similar color palette and design of the individual components in the screens. The creation of the icon for our app and its naming, BookIT, was done by MS. After its proposal, the rest of the members agreed on it.

Having a good head start, focus was set on the development. Presentations were shared as a Google Slides file, giving the ability to all members for real-time editing. The organization of the meetings was done depending on the needs of the individual members and their time schedule to align and accommodate.

Jira was used as an organizational tool for the delegation of tasks inside the group. Providing a clear view of the tasks currently in progress as well as their assigned group members.

The presentations during the supervision meetings were structured to highlight the previous feedback and explain what our solutions as well as thoughts were on the matter. Proving to be a beneficial decision as it sparked feedback and comments on the parts our members were interested in.

Background

At the premises of both Mälardalen University's campuses, there are a series of rooms which may be used by students for organizing study sessions, and for staff members to organize lab sessions, additional hours, meetings et cetera. An issue which may arise is where certain rooms may get more bookings than others. Precisely, the recommendation system needs to be fair, with the goal of making sure that all rooms are booked equally. Furthermore, another issue is the lack of personalized allocation for rooms. Meaning, there is not a recommendation system for students according to their specific needs. The goal of this project is to allow rooms to be booked as evenly as possible, and for students to receive the best room according to their preferences.

Currently, there exists a room booking system called KronoX. It is used by universities in Sweden for room bookings. At the moment, KronoX does not have smart allocation algorithms implemented (KronoX Web, n.d.; KronoX, n.d.). Furthermore, with KronoX, it is not possible to create events, which other users may join. Also, the user interface of Kronox for universities is not intuitive, and would take time to understand it properly (KronoX Web, n.d.). These are the issues which we are attempting to tackle with this project.

Methods

Jetpack Compose Application

The team worked together on crafting the app's design through Figma thanks to its easy-to-use interface that made creating designs smooth. Responsibility was shared among the team members with each one taking charge of at least one app page which encompassed Events Dashboard, Profile, Map, Search Rooms, and Room Details. By splitting up the tasks each person was able to put in a fair share of work and it also opened the door for different kinds of creative ideas to flow in.

In deciding on the theme for the app the team went with a lively set of colors focusing on various tones of orange. These colors are identified by the following hex codes: #FF8C42 #FFBD92 #FF7BD1 and #FFEEE3. The reason behind picking this specific color scheme was to create an atmosphere that feels welcoming full of energy and warmth. This vibe is carried throughout every page to ensure everything looks consistent. Moreover, it aligns perfectly with the colors found in the Malardalen University logo.

After wrapping up the design work in Figma it's time to dive into the development phase. We chose Jetpack Compose for our coding needs; it's a fresh and very popular tool for Kotlin that makes building user interfaces a breeze. In light of the tasks that had piled up each member of the team took on the job of bringing at least one page of the user interface to life. They made sure it looked and worked just as we had first imagined. By adopting this methodical strategy, we were able to turn our designs into a working app without wasting any time.

In the time when we were developing our project the way everyone in the team worked together was really important for keeping things straight and moving fast. Git was our go-to for keeping track of changes in the project. It made it easy for all of us to work at the same time without messing up the project and making sure everything stayed in order. By adopting this method every person's input was seamlessly woven into a unified app.

The application allows users to book rooms, search for rooms using a map or the search page, join and create events. The main pages are available on the navbar on the bottom and the detail pages are available by clicking on buttons on the main pages. The profile page has its own button on the top right. At first there was an option not to log in, but it was decided that logging in is mandatory from the start to avoid being asked to log in on most pages.

The community board displays the events available and when you click on it, we have more details about it. There is an option to join on the details page. The map allows you to choose a campus and a floor, we can navigate through it to see

the rooms names. The search page allows you to search for a room at a given time with different filters like the number of people, the campus ... The algorithm will give you the room that fits the most as the first result.

The user can also access the reservation page, the profile and other features on other pages. You can book a room by searching for it on the search page and then by making a reservation. That reservation can be found on the reservation page. When you click on the reservation, you can see the details of that reservation and cancel it if needed. A map section also displays the room location on the same page. When the user creates an event, it can be canceled on the reservation details page and there you can access the event details too. The user can also join events by going to the community board page and by clicking the event to go to the details. There is a button to join or leave the event. The user also can access the profile page to see or change some settings. This page is available by clicking on the top right icon.

The admin page is only available if the user is an admin. Admin has a special role and gives you access to a special page, the admin page. This page gives you access to the rooms and makes you able to change the information, hide a room from the users or delete a room.

Node.js backend

The application's backend was written in Express.js, a framework built for Node.js. Express.js provides an easy way of setting up an API, with an intuitive way of writing routes, and all of the accompanying methods like middleware and routers (Express.js, n.d.). All team members had previous knowledge of JavaScript and Node.js, therefore opting to utilize our already present skillset.

The backend is accompanied with *Prisma* for detailed data reading and manipulation. This library was chosen due to its ease of use, and the fact that it is not required to write any SQL in the code (Prisma, n.d.).

Using Prisma, a database schema can be written, after which a database migration can be created using Prisma's powerful command-line interface (CLI). This offers easy updates to the database, with an added downside of data loss. To address that specific issue, we prepared queries to insert test data via a DBMS (Database Management System). For the purposes of our project, we were using DBeaver. Furthermore, Prisma offers out-of-the-box validation features for database entities, which provides another layer of security for the application. It prevents unwanted data access and manipulation using external API testing tools, such as Postman or Insomnia.

For authentication and authorization, *Passport* was used. It is an external library which provides authentication and authorization middleware, and a JWT strategy for applying authorization rules using a decoded JWT payload (Auth0, n.d.).

As per the security standard, the Bcrypt hashing function was used for hashing passwords, with 12 rounds of hashing taking place. When a user is registering, a role is assigned depending on the user's email specified.

For storing sensitive data, the "*dotenv*" library is used. It provides a way to access environment variables from the .env file, which should not be published. It contains values for the database URL, JWT secret and the admin email.

The backend is configured as such that it can only process HTTPS requests with a self-signed certificate for development uses. In an application such as this, data integrity must be ensured so that data is not tampered with, or that any data is not compromised.

The architectural choice for the backend code was to fully utilize separation of concerns. Express.js does not provide a default architecture for its projects, thus implementing our own style. The business logic was separated into services, where each service contains data manipulation methods for a particular entity. In addition, there is a JWT service, which contains methods for signing and verification of tokens, and an authentication service, which contains methods for logging in, registration, and password changing. The service methods are used in routers, which contain endpoints for a particular entity or if the router is for authentication endpoints.

Results

Map development

The map feature was designed to accommodate the styling and functionalities required for user interactions. As there were two possible libraries to be utilized, Mapbox and Google Maps (Mapbox, n.d.). Further research was needed. One of the criteria for the conclusion for the decision of the wanted map library was the cost. As Mapbox offered an initial larger free use capability, it was later discovered that computational units depending on the addition of labels were also charged, proving to be a limitation. A workaround was provided by converting all the individual labels into one GeoJSON file to require less computational power as well as limiting the units to just one interaction; therefore, the MapBox Studio is just recording one unit even though the amount was larger.

The MapBox studio offered dataset creation of vector points representing individual real-life coordinates. The interface provided by MapBox was easy to understand and to use, while still needing an extensive amount of time to mark all the points of interest in the dataset. One of the possible solutions was the upload of a rasterized image; while in theory that would work, the Mapbox Studio didn't offer such capabilities.

The conversion of the datasets to tilesets was a necessary task, as Mapbox styles could only utilize tilesets. This proved to be an issue as Mapbox had simplification algorithms, which translated the vector datasets into simpler polygon structures from the ones marked. The solution for mitigating this issue was the MapBox Python library, which needed to set custom zoom levels that weren't able to be manipulated in the MapBox Studio interface (Mapbox Android API, n.d.). The first task was converting the datasets GeoJSON files into LDGeoJSON files to be usable in the Python library. The creation of the dataset source and the conversion to tilesets were all done through commands in the terminal. The creation of a unique recipe file was also mandatory for each tileset, providing its structure as well as the custom zoom levels.

JWT authentication

For the purpose of authentication and authorization, JSON Web Token was used. JWT is a compact and secure way of transmitting information between different parties. It is an immensely reliable technology, since the information is digitally signed using either a secret key (HMAC algorithm), or a public/private key pair using RSA for example. Compared to server-side sessions, JWT takes less toll on the server, and is stateless in nature.

When a request is sent from the client, there will be an Authorization header included, which contains the JWT. By including the token in the request, the user is able to access protected routes.

JWT consists of 3 parts which are separated by full stops. They are as follow:

- Header
- Payload
- Signature

A header consists of 2 parts. It specifies the type of token, and the algorithm which is used for signing. It looks as follows:

```
{
 "alg": "HS256",
 "typ": "JWT"
}
```

The second part is the payload, which further consists of claims. Claims are statements which are transferred using the JWT. There are three types of claims: *registered, public* and *private* claims.

Public claims can be defined by anyone using JWTs. In the context of BookIT, public claims in the token are the user id, email, first name, last name and the user's role. For security reasons, it is not recommended to have the hashed password as a public claim, even though the hashing function cannot be determined from the hash provided. Private claims are custom claims which are used to share information between multiple parties that agree on using them. In our application, we do not use such claims. Finally, registered claims are predefined, which may include claims such as expiration time. Its value is a time in the Unix epoch, at which the token will expire, and when the user must log in again.

In the application, the token is stored in a ViewModel. A ViewModel in Jetpack Compose provides a wrapper for the entire application which is initialized when the MainActivity is created. Inside of it, the global state may be kept. In BookIT, the token is stored, which is further decoded and the decoded public claims are stored as user data. It is decoded using a method from the JwtService class.

Community Screen

The community screen serves as an event board and news page for all app users, enabling them to stay informed about current university events. Ranging from study groups to presentations and fairs. It has a structure compiled of cards, each representing one event, with the details provided, time, location, brief description of the event, and a title. The event cards also have an indicator based on the room type, associated with a color.

The CreateEventDialog composable is used for the creation of new events with the following interactive features:

A dropdown menu to select an available reservation, ensuring that events are associated with specific rooms and time slots. Text fields for entering the event's title and description, each with character limits (20 for the title and 350 for the description). Real-time feedback on the character count is displayed to guide the user. Confirm and cancel buttons for submitting or discarding the event creation process.

The dialog ensures that the user cannot create an event without providing all required details, including a reservation, title, and description. On submission, the event details are sent to the API, and the events list is refreshed.

Search Screen

The Search Screen is one of the key components of the app, it enables users to find and filter rooms easily. Implemented using Jetpack Compose, the screen offers a very responsive and user-friendly design made to meet the needs of students and staff.

Data is fetched when the screen is loaded, utilizing Retrofit to retrieve room details from the backend. The fetched data is processed into a structured format and displayed dynamically. State variables like detailsList, isLoading, and errorMessage ensure the interface updates seamlessly based on the current status, whether loading, successful, or encountering an error.

The Search Bar allows users to filter results by typing queries, with instant updates to the displayed room list. The results are presented as Search Details Cards, showcasing key attributes such as room title, type, and availability. Each card is color-coded for clarity, making it easy to differentiate between lecture halls, labs, and study rooms at a glance.

A Current Date and Time Card adds contextual information, displaying the current date and time in an appealing design. A Floating Action Button (FAB) at the bottom of the screen serves as a call-to-action, encouraging users to perform a room search effortlessly.

To enhance the user experience, loading indicators provide feedback during data fetching, while error messages clearly communicate issues when they arise. The screen's design follows Material Design principles, incorporating a gradient background, consistent typography, and intuitive layouts.

This screen streamlines the room search process, offering a quick and intuitive way to find available spaces. Its scalable design ensures it can accommodate future features, such as advanced filters or personalized recommendations, making it a valuable part of the university's booking system.

Reservations Page

The reservation screen is a feature available for all users. This screen allows the user to have a look at his reservations (booked rooms). This page only displays the reservations of the logged in user. When the user clicks on a reservation, the reservation details screen is opened, and the user can have more details about the reservation and on the top, a map section shows where the room is to make it easy for the user. The user also can cancel the reservation if wanted.

The reservations are fetched through a specific API endpoint. We get every reservation of the current logged in user bases on the id. Then it is displayed as cards to the page. The page is scrollable if there are too many reservations to see on the screen. A new request is made to the database every time the page is loaded to have real-time information. So, with that, the user has no way to access the reservations of the other users.

When the user clicks on a reservation, it redirects to the event details page. This page fetches only one reservation by its id and displays all the information of that reservation. It allows the user to cancel the reservations by clicking a red button. When a reservation is linked to an event, an additional button is displayed to go to

the event details. When an event reservation is canceled, the event is deleted, and the reservation is also deleted on cascade automatically by the database. All of this is done through API calls with different endpoints.

Events and Reservations

The event screen that we named community board is the main page of the application. It is the screen that is displayed first when the application is opened. This page displays public events available to all users.

There is a button to create an event with the possibility to customize everything like the title, the description and the reservation. The event is linked to a reservation that is already done by the user. In other words, the user needs to make a reservation first before creating an event. So, to do that, multiple calls to the API are made to first create a reservation and then link it to an event. The events are on one table and the reservations on another one. An event is always linked to a reservation. This means that an event relies on a reservation and this event can be canceled like a reservation. The fact that a reservation is linked to an event is specified in the reservation details screen.

Through the event details page, every user can join some events if there is room left. They also can leave the event if they want. This is done through a separate table in the database to handle the users in each event. This table is called the event participant's table. There is a verification done in the backend to see if an event is full and to restrict users from joining a full event.

Profile Page

The profile page can be accessed by pressing the person icon in the top right-hand corner of the screen. Its main purpose is management of user preferences for the application. In addition, it provides the option for the user to change their password, under the condition that they already know their current password.

The page is split into three sections:

- Account
- Notifications
- Preferences

In the Account section, the user can see the email with which they had registered their account. Below the email box, the user may find a button, with an icon of a triangle. on the right. Said button may be pressed, after which a modal with a form will be opened. This form provides the user with fields to enter the current password, the new password, and to repeat the new password. The user will be promptly notified whether their request has been successful or not.

In the Notifications sections, the user may opt in to receive notifications about their pending reservations. By pressing the switch, the user chooses whether they would like to be notified about reservations.

In the Preferences reservation, the user chooses their campus, which influences the allocation algorithm by providing the user with rooms from their desired campus.

Login/Signup Pages

When the user wishes to open an account to use BookIT, they ought to visit the Signup page. It can be accessed from the login page, the first page that is shown after the app is opened. The user provides their full name, mdu email, and password, which must be confirmed. The password must be at least 8 characters long, and it must contain a special character and a capital letter. The topic for discussion within the group was whether to use the full name during registration, or a username. It was decided that, due to the fact that there is more space for faking a username, we would use full name. We decided to assign roles based on the email domain name. Student is given for student.mdu emails, admin for admin.mdu users and plain mdu users are for professors and other staff.

As for the Login page, the user enters their email and password. After filling out both forms, the data will be sent to the backend, after which a JWT will be returned, and further stored in the ViewModel. Then, the user will be redirected to the community screen. Depending on the role, the user will have different app functionality that was explained in methods paragraphs. If there was some kind of error while logging in, the error message will be shown giving insight into the error that occurred and a way forward for potential fixes to resolve the error.

Admin Page

The Admin Page is a feature designed for users with administrative privileges, accessible via the navigation bar when logged in as an admin. The interface provides functionalities such as modifying room availability, adding new rooms, updating details of existing ones, and deleting rooms or reservations. The page uses state variables to manage loading, error messages, and dynamic updates to the room list. The UI features a scaffold structure, a loading spinner during data fetches, and displays either error messages or a list of room cards rendered using the RoomCardAdmin composable, enabling room management actions. A floating action button at the bottom opens a modal dialog for adding or editing room details. Changes are reflected immediately in the displayed list.

Calendar Pages

When booking, there is a need to filter and reserve bookings by time. Hence, the calendar pages. Since the flow of the app provides two ways to select the time of bookings, two calendar pages had to be made. One for filtering and one for confirming bookings. Both are similar but provide different functionalities to accommodate the user's needs.

When filtering rooms by time, there is no need to indicate which are occupied, since that will be portrayed through the search results by only showing vacancies.

However, when a room has been selected without applying a time filter, the booking must be finalized by selecting when the room should be booked. The view of the calendar page on a selected room can compare the time slots to earlier reservations by fetching reservations for said room. This way users get an overview of the room's schedule.

To know if a time slot is free to book there is a fetch to the database, using a route in the API, when navigating to the calendar page. When a time slot has been reserved it will be colored red to indicate unavailability and its button will be disabled. This prevents conflicting reservations and ensures that the user is the sole owner of the reservation. It is impossible to overlap bookings since all affected time slots will be rendered unavailable.

The calendar limits students to make reservations up to one week in advance, meanwhile, staff can book rooms two weeks in advance. In addition, there is a limit of two concurrent bookings. The role is recognized through the decrypted JWT which is passed through the viewmodel. The reason behind these restrictions is to inhibit students from occupying all rooms. This encourages students to be considerate when booking rooms and not book rooms they do not need, ensuring everyone can utilize the study rooms that the University offers.

Room Details Page

Navigating to the room details greets the user with information about the selected room. The info gets retrieved from the backend using the room ID passed from the search page. The details contain the name, capacity, type, location and description of the room. Additionally, it provides a small container with a map to guide the user to the room.

After the user has found a suitable room there is a button that will either book the room or prompt the user to select a time depending on whether the user applied a time filter. If the user has already made two reservations the button will instead display that max bookings have been exceeded.

If the user wants to search for another room there is a button to pop the stack to navigate back to the search page. Once a room has been booked the user will be navigated to the reservation page to view their reserved rooms.

Discussion

Feedback from the presentation

During the presentation, we had feedback from the other students as well as from the teachers. Some things were said multiple times and here are the main things that were said:

- The event creation should not be available if there is no reservation to avoid starting the creation and finding out later that you can't create an event.
- Joined events should display something on the page to know the event has been joined without needing to click on the event.
- The search page should be more intuitive, maybe there should not be a button to search, and it should update every time we change the filters
- The map page looks good, and it is a very good idea to find rooms easily
- The admin page could have a way to search for specific rooms, but it still is easy to understand and use.
- The global flow of the app is good and intuitive; some things need improvement like what was said in the earlier feedback. However, in a global way the tool felt useful and understandable.

Further improvements

At the moment, the application has the basic allocation algorithm implemented, but its formula would need time to be perfected. To further improve it, additional suggestion criteria could be implemented.

Furthermore, according to feedback, the user experience is not fully satisfactory, and requires further development. From the side of the team, a more thorough testing should have been done, where not all issues were detected. An example of an issue would be the one where the JWT from the previous user would reflect on the currently logged in user inside of the application. This issue must be addressed and resolved, due to the fact that the app would be rendered unusable, and would imply a security breach, where the currently logged in user is free to modify the previous user's reservations.

Acknowledgments

We would like to express our sincere gratitude to our professors Afshin Ameri, Philip Lindstedt, and Leo Hatvani for their guidance, constructive feedback, and continuous support throughout this project. Their expertise and insights on system design, allocation algorithms, and usability were invaluable to our progress.

We also thank our fellow students from the School of Innovation, Design, and Engineering at Mälardalen University for their helpful comments and feedback during supervision meetings, which greatly contributed to improving the functionality and user experience of our application.

Finally, we acknowledge the dedicated efforts of all project members, Basar Carovac, Benjamin Marguin, Danjel Frrokaj, Felix Åhlén, Ivan Blazanovic, and Momir Savic for their collaboration, creativity, and commitment during the design, development, and testing phases of the Smart Room Booking System.

Bibliography

1. Auth0. (n.d.). *JSON web tokens introduction*. https://jwt.io/introduction
2. Bowman, M., Debray, S. K., & Peterson, L. L. (1993). Reasoning about naming systems. *ACM Transactions on Programming Languages and Systems*, 15(5), 795–825. https://doi.org/10.1145/161468.161471
3. Ding, W., & Marchionini, G. (1997). *A study on video browsing strategies* (Technical report). University of Maryland at College Park.
4. Express.js. (n.d.). *Express 4.x API reference*. https://expressjs.com/en/api.html
5. Google. (n.d.). *Google Slides*. https://workspace.google.com/products/slides/
6. KronoX. (n.d.). *Framtidens schemaläggningssystem*. https://kronox.se/
7. KronoX Web. (n.d.). https://kronoxprod02.mdu.se/
8. Mapbox. (n.d.). *Mapbox documentation*. https://docs.mapbox.com/
9. Mapbox Android API. (n.d.). https://docs.mapbox.com/android/maps/api/11.9.0/
10. Prisma. (n.d.). *Getting started with Prisma*. https://www.prisma.io/docs/getting-started