

HKDSE 2024 ICT SCHOOL-BASED ASSESSMENT

Venue Booking System

6C19 Wong Ka Lok

Carmel Pak U Secondary School

Table of Contents

TABLE OF CONTENTS.....	1
OBJECTIVE	2
ANALYST & FURTHER THOUGHTS.....	3
DESIGN & IMPLEMENTATION	4
SPRINT 1 – START OF DEVELOPMENT (4 DAYS)	4
SPRINT 2 – IMPLEMENT OF DATABASE & BASIC API ENDPOINTS (4 DAYS)	5
SPRINT 3 – REWORK ON UI (5 DAYS)	7
SPRINT 4 – FINISHING UP WHOLE SYSTEM (7 DAYS)	8
COST SUMMARY	11
BASIC SETUP UP GUIDE.....	12
ENVIRONMENTS.....	12
SERVER	12
USER GUIDE.....	12
LOGIN.....	12
CHECKING YOUR OWN BOOKINGS.....	13
CHECKING ALL BOOKINGS.....	14
CREATING A NEW BOOKING	16
CHECKING AND UPDATE YOUR INFORMATION	18
CONCLUSION	19
REFERENCE	19
PUBLIC.....	19
PRIVATE	19
APPENDIX.....	20
@/prisma/schema.prisma.....	20
@/prisma/seed.js	21
@/pages/api/login.js	27
@/pages/api/staff.js	27
@/pages/api/venue.js.....	28
@/pages/api/booking.js	28
@/pages/api/booking/[id].js.....	30
@/app/layout.jsx	30
@/app/head.jsx.....	30
@/app/page.jsx.....	31
@/app/login/head.jsx	33
@/app/login/page.jsx	34
@/app/dashboard/head.jsx.....	36
@/app/dashboard/page.jsx.....	36
@/components/dashboard/Calendar.jsx	39
@/components/dashboard/Overview.jsx.....	41
@/components/dashboard/Profile.jsx	43
@/components/global/BookingInfoModal.jsx	44
@/components/global/CreateBookingModal.jsx.....	47
@/components/global/CustomAppBar.jsx	51

Objective

CPU School is going to set up a booking system for staff to book the venue in school days. School days may include from Monday to Saturday, except for public holidays and Sundays. Each venue can only be booked by a user simultaneously.

The following venues are available:

Cyber Hub, Inno Lab, STEAM Cave, Idea Lounge, MMLC, GP Room, Art Room, Music Room, Lecture Theatre

Bookings are divided into periods according to the following timetable.

Period	P1	P2	P3	P4	P5	P6	Lunch	P7	P8	P9	After School
Time	8:25 - 9:05	9:05 - 9:45	9:45 - 10:25	10:40 - 11:20	11:20 - 12:00	12:00 - 12:40	12:40 - 13:45	13:45 - 14:25	14:25 - 15:00	15:00 - 15:35	15:35 - 18:00

Any user can book at most 1 venue at any given day and at most 4 bookings at any given month. The booking system should also be able to generate venue usage report for the administrator. During the data collection stage, the booking information will be input into the program, as shown in the following example:

- Username (staff initial, e.g. YA, KA ...)
- Venue name (1B, F5 ...)
- Purpose (ICT, cell group ...)

Analyst & Further Thoughts

As what the objective has mentioned, we are going to build up a system about booking venue at school. Our target audience is staff only, we need to prevent student from using this system, so a login system is required.

With my previous experience about full stack development, instead of building the system with C/C++ with CLI only, I decide to use some framework with better GUI implementation and break the system into frontend and backend.

The frontend of the web service is about different interactive elements like records display or submission. And the backend will handle all the data storage and query.

Here is the first thought on the structure of the frontend:

- Login Page
- Home Page
- Dashboard
 - Overview
 - Calendar
 - Profile

There are different GUI supported framework for web development, and the most famous is the React.js with JavaScript and TypeScript supported.

React is a framework that does server-side rendering which all the rendering process will only occur on the server side of the web server, there will be no JavaScript to be loaded on the client browser. It is more efficient and secure compared to the old day when some of the script need to be loaded on the browser and take the risk of injection.

The framework I chosen is one of the mutations from React, Next.js, which implemented API service that allows me to build the backend service at the same time.

Design & Implementation

There are current many new development methods, like Waterfall, RAD and Agile. Since I am doing a web development and with my previous experience, RAD and agile are more suitable for my situation, so I choose agile development methodology.

Agile is usually applied in app and web development and whole development process can be very flexible. All the tasks are broken into small pieces to be easily managed and rearranged. We do not need to go back to the very first stage as Waterfall when we experience any major changes. Although the cost is not totally fixed and be easily estimated, we are much comfortable with the flexibility of this method.

A normal development is around 6 to 8 sprints where each sprint is about 1 to 4 weeks long. As we are not required to submit the “Testing and Evaluation” part and it is only a small-scale development, the total sprints required can be reduced and the length of each sprint can be shortened.

Although the part of “Testing and Evaluation” is not required, there will still be some unit test to make sure every function is under control.

The following will be recorded as sprint review after each sprint.

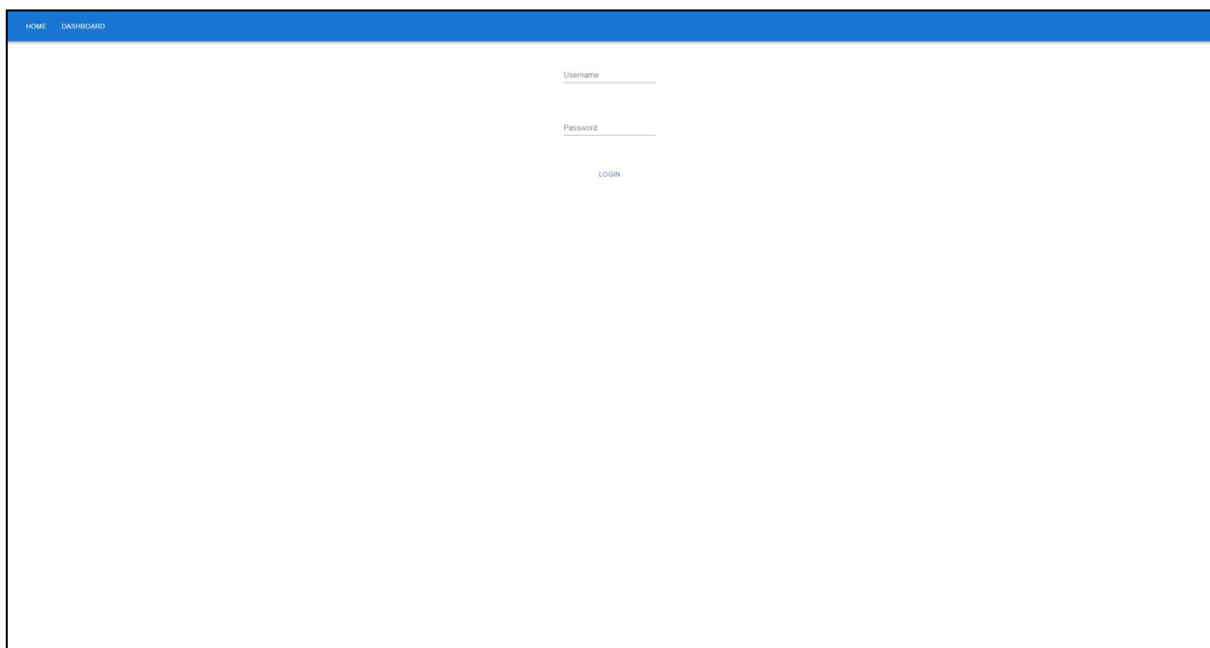
Sprint 1 – Start of Development (4 days)

For the first sprint, I decide to quickly build up the skeleton of the page.

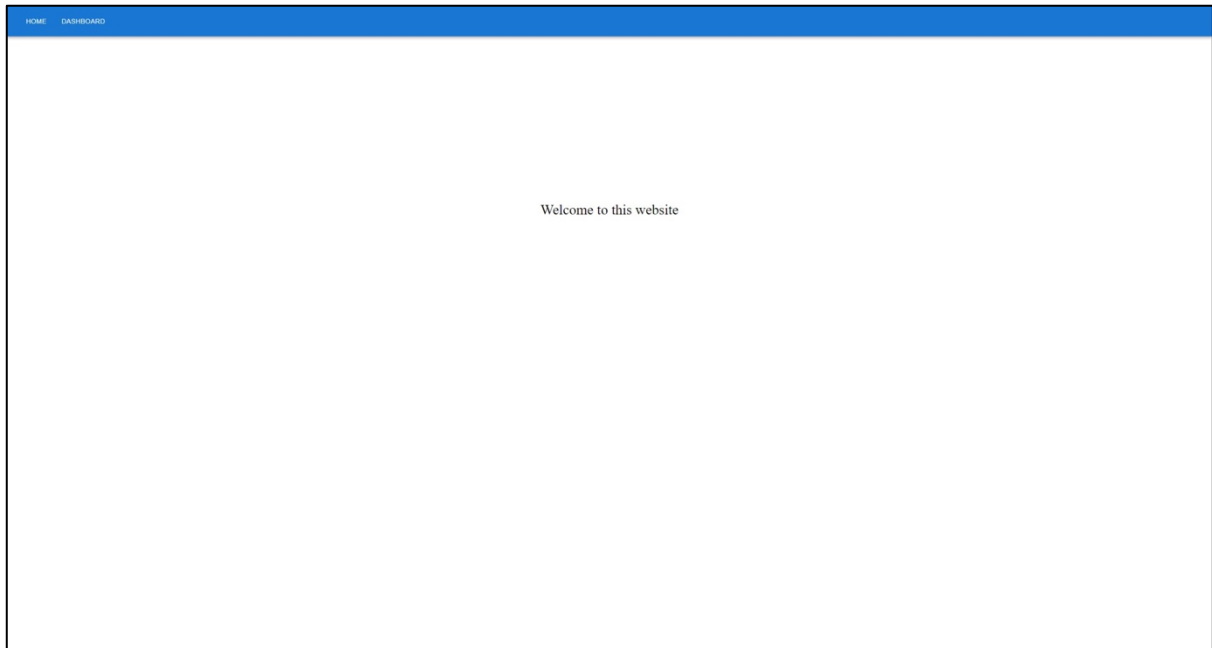
As I have dropped quite a lot of time on other aspect of technology.

Time is required to re-pick-up the skills of web development.

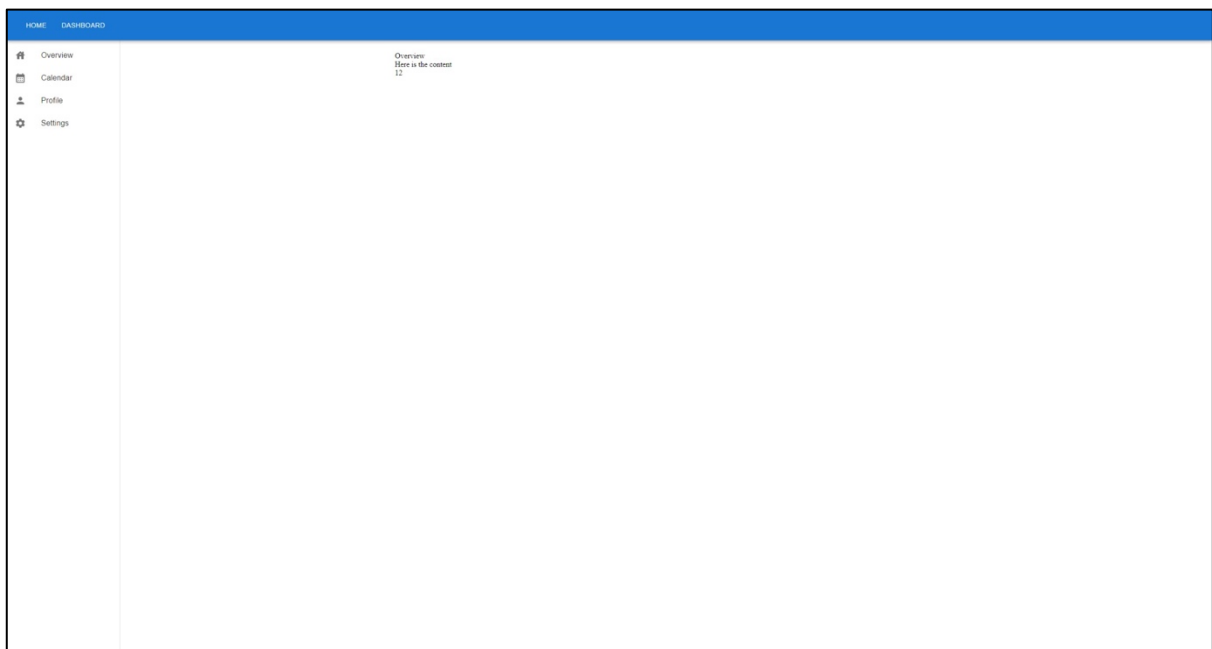
I took around 2 to 3 days to read the documentation to get myself up to date for setting up a simple web page with the new update of the framework I am using.



The 1st version of the login page.



The 1st version of home page.



The 1st version of the dashboard without any split screen

Then the first stage of the development is completed.

Sprint 2 – Implement of Database & Basic API Endpoints (4 days)

As a complete booking system, a static storage is required for storing different type of data. There is different method of storing data, like cache memory, text file, or even a database.

Local Memory is not preferred as we are having a complete API and text file is not convenient for query. Also, this is a small-scale web service, some of the common database like MySQL or MongoDB is not the best choice for this repository as they need to be separately hosting the database server.

And finally, we come up with SQLite as it stores data on a well-structured database file, we can access it offline. Therefore, it is the most suitable database for this project.

Then we need to consider what to be stored on the database.

There should be at least a table about bookings as what is the web server serving for.

If there is login, a table of user is required to identify users.

Besides, I would like to modularize some of the fields, a table of venues is needed to dynamically import the available venue.

Thus, we got three different tables on the database, and we further think about the fields of data to be stored.

Then we got a list below:

- Booking
 - Id (Unique)
 - Booking Date
 - Booking Period
 - Reason / Purpose
 - Staff
 - Venue
- Staff
 - Id (Unique)
 - Username
 - Email (Unique)
 - Password
- Venue
 - Id (Unique)
 - Venue Name

And we convert into the schema of the database ([Appendix: @/prisma/schema.prisma](#)), as Prisma is the default client used to locally query the database.

After implementing the database, we need to setup a bridge to connect frontend and backend which is called API (Application Programming Interface). It provides an interface for client to access the data located on database.

Basic endpoints are required like booking, staff and venue, these endpoints should be easily access so it should be something like `/api/staff` or `/api/boking/[id]`. And have the following endpoints:

[/api/staff](#)

[/api/venue](#)

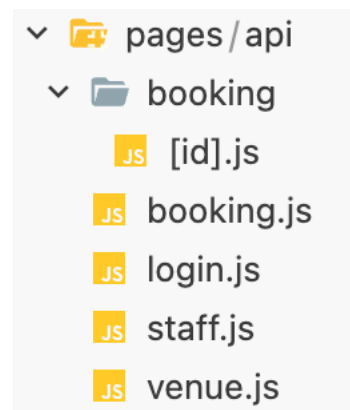
[/api/login](#)

[/api/booking](#)

[/api/booking/\[id\]](#)

All the endpoints are tested with an API platform, Postman.

And the way it function is written down as comments beside the related code.



Sprint 3 – Rework on UI (5 days)

We have got the idea of what to be shown to client, we turn the side back to frontend development. And we also realize the UI suitable for everyone, so we decide to rebuild most of the UI components and finishing up the remain screen on dashboard.

The new UI can be found on [User Guide](#) but only UI has been finished on this stage.

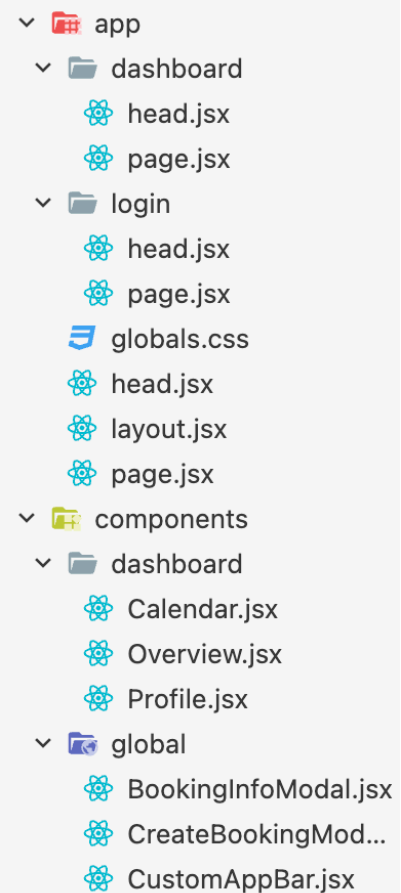
As the login page is too large but the field is too small, it is not a good practice of UX, the client will find difficulties when trying to login. Therefore, I separate the full login page from a part of the home screen, then enlarge and center the components inside the page, applied with some themed text field and button ([Appendix: @/app/login/page.jsx](#)).

Then rework on the App bar to add a logout button for the user to logout the system ([Appendix: @/components/global/CustomAppBar.jsx](#)).

Firstly, I have no idea what to put on the Home page, but when the developments continue, I think it is good to have a screen that display all of user's bookings so I added a list to display all the user's bookings ([Appendix: @/app/page.jsx](#)).

Also finished up building the three-screen located ([Appendix: @/components/dashboard](#)) on dashboard. However, the UI tool I am using ([Material UI](#)) does not support calendar view, so I go and found an alternative package ([FullCalendar](#)). List UI is used to be an alternative way to view all of the current bookings and setup a simple Profile screen for user to check and update their profile.

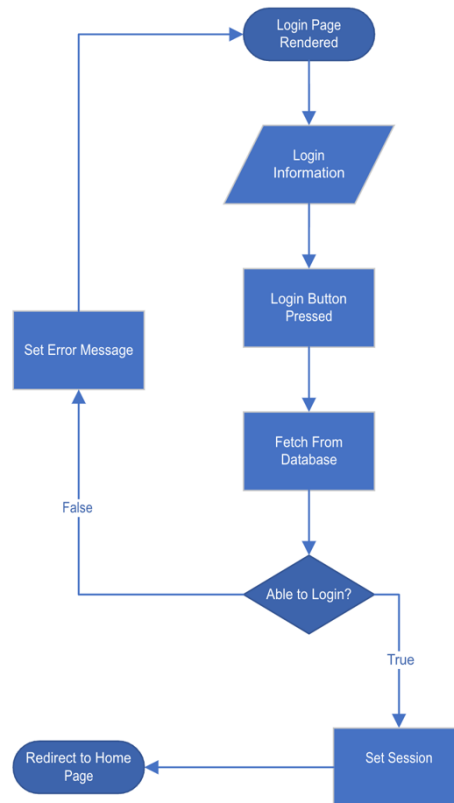
We experienced some issues of components do not align to the screen, that cost us quite a lot of time to debug and finally finishing up our third sprint.



Sprint 4 – Finishing Up whole System (7 days)

We now have both frontend UI and backend database with API. The final stage will be connecting all these together to get a truly functional system.

First, we consider the way to login. This is the flow of login.



We must validate and verify the input, so I added error message on fail. The api endpoint will do both validation and verification at the same time ([Appendix: @/page/api/login.js](#))

CPU Venue Booking System

Log in

Credentials cannot be empty!

Credentials cannot be empty!

LOG IN

You are not able to login without entering any credentials.

You are not able to login without correct data input.

However, we found difficulties while storing session after login, server always failed to get the user information and return user back to login page. Fortunately, we discovered the solution to easily fix the problem by accessing the local cache of browser as session is only temporary variable, we can store the session on localStorage and access it at any time ([Appendix: @/app/login/page.jsx](#)).

Then we put our focus on creating new booking. As the same as login, validation is required, we put our new created booking to backend and do checking.

You are not able to submit empty booking.

HOME

DASHBOARD

LOGOUT

Overview

Calendar

Profile

Overview of all bookings

CREATE NEW BOOKING

Venue	Teacher
STEAM Cave	Admin
GP Room	Admin
STEAM Cave	User
Inno Lab	User

Create New Booking

Venue*

Inno Lab

Date

09/23/2023

Period*

3rd Period

Reason*

Try to repeat the booking

Booking already exists!

CANCEL

SUBMIT

Rows per page: 100

1-4 of 4

You are not able to repeat book the same venue at the same time.

HOME

DASHBOARD

LOGOUT

Overview

Calendar

Profile

Overview of all bookings

CREATE NEW BOOKING

Venue	Teacher
STEAM Cave	Admin
GP Room	Admin
STEAM Cave	User
Inno Lab	User
Inno Lab	User
MMLC	User

Create New Booking

Venue*

Idea Lounge

Date

09/23/2023

Period*

2nd Period

Reason*

asdfasdf

You are not authorized to create more than 4 bookings!

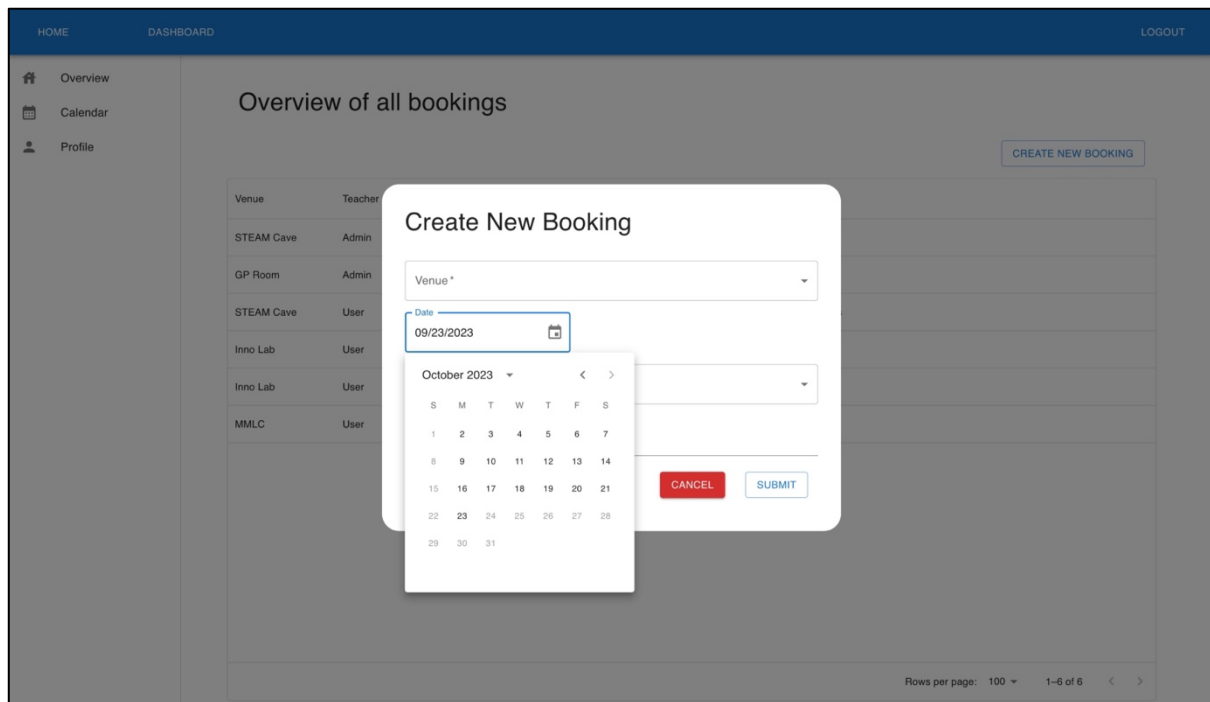
CANCEL

SUBMIT

Rows per page: 100

1-6 of 6

You are only allowed to have four booking with in a month.



You are only allowed to book on weekdays and not more than a month.

And the choice of venue is dynamically banded to </api/venue>

Next, is about deleting bookings, the delete button is located on the booking information modal. And we just need to bind it to [/api/booking/\[id\]](/api/booking/[id]) with the method “DELETE”.

After that, the most important part is the list of bookings to be display, and it required to be filtered with staff for the home page and get all bookings for the dashboard on </api/booking>.

We finalize the database with some preset values for demonstration. ([Appendix: @/prisma/seed.js](#)) Which can be called into database by simply typing the following command in the terminal of your root of the repository:

```
cd src
npx prisma db seed
```

The actual workload of this sprint is slightly higher than expectation, mainly experienced that the data is not well formatted or not able to query from the backend. Turns out that I used one week for this sprint, we should break down in to two sprints during sprint planning.

Cost Summary

We totally spend around 4 sprint (20 days) with around 8 working hours a day to finished to web server. Referring to the some of the data (<https://arc.dev/freelance-developer-rates/full-stack>), the average hourly salary of a full stack developer is about 60-80USD. Therefore, the average cost of this production cycle will be around 11200USD (about 87589HKD). If we hire a junior developer, it is also about 30000-50000HKD for salary.

Basic Setup Up Guide

Since this is a webserver instead of a desktop compiled application, so before using the system, we need to setup the environment and bootup the system.

Environments

Make sure you have installed [node.js with npm](#) before you start to use this project. Nodejs with version 18.14.x and npm with version 9.6.x is preferred.

Server

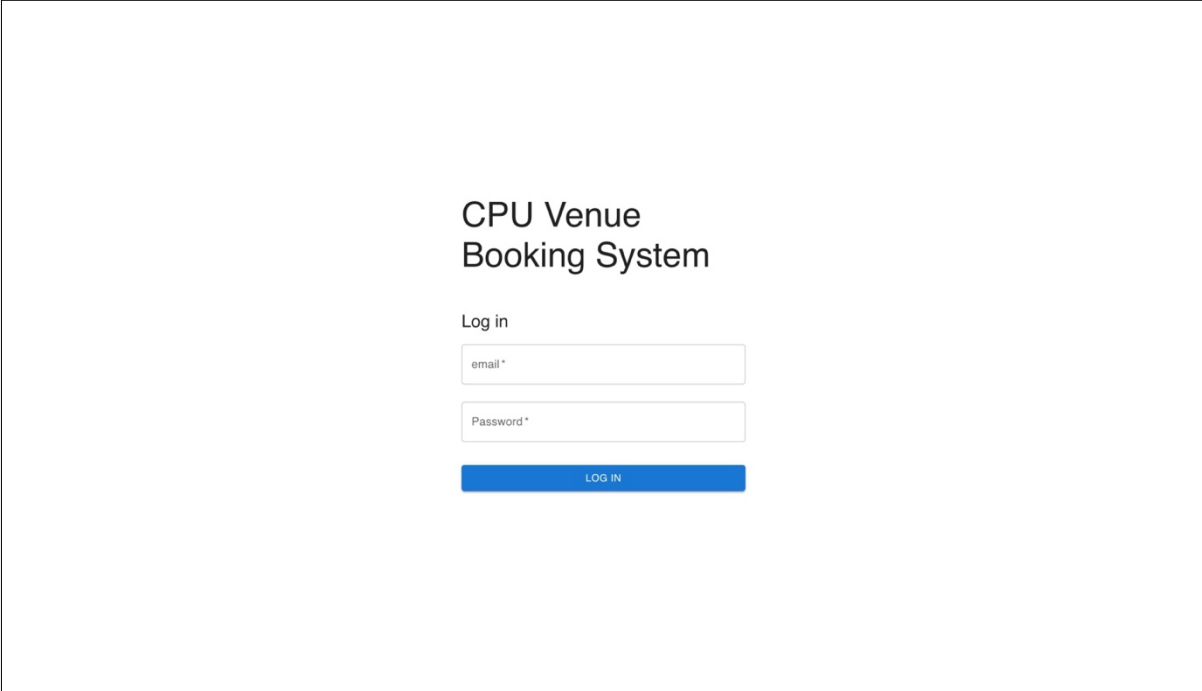
Run the following in the terminal with the root of the repository.

```
cd src
npm install
npm install --save-dev
npm run build
npm run start
```

After all setup sequence has completed, open the website <http://localhost:3000> in your browser.

User Guide

Login



CPU Venue
Booking System

Log in

email *

Password *

LOG IN

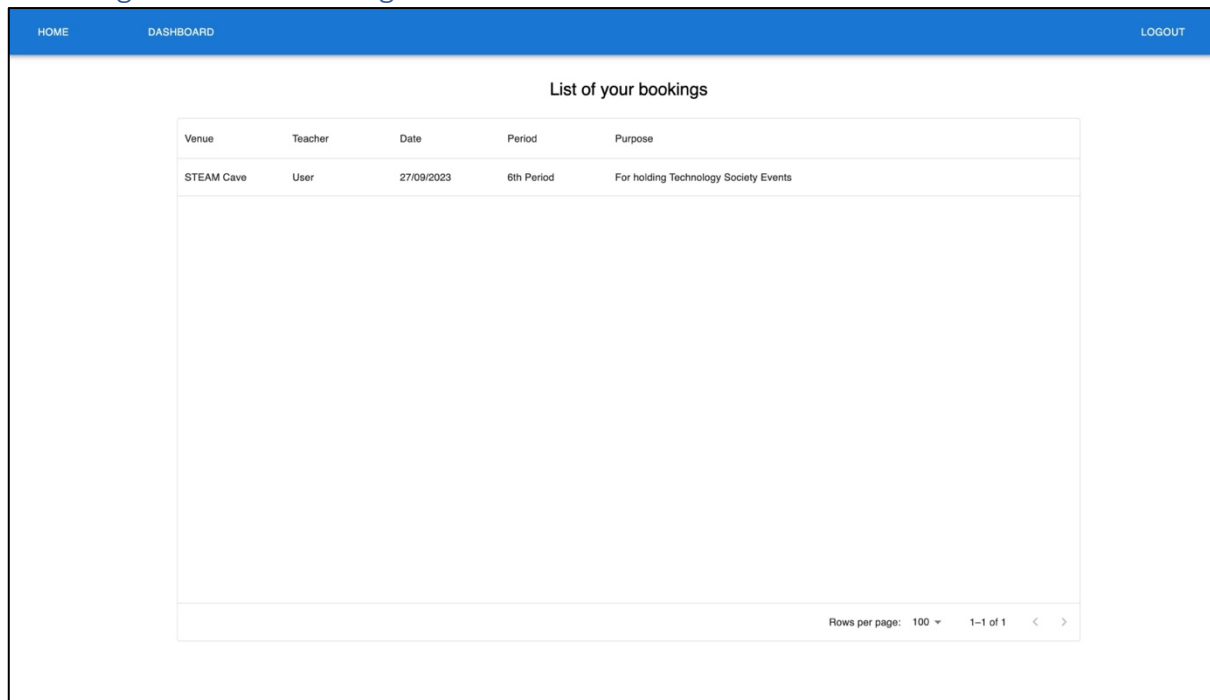
This is the first page you will see when you first launch the system.

It is a login page located at <http://localhost:3000/login> that allows you to access the system with permission.

You are not able to access the system without logging in the system.

It will redirect you back to the login page.

Checking Your Own Bookings

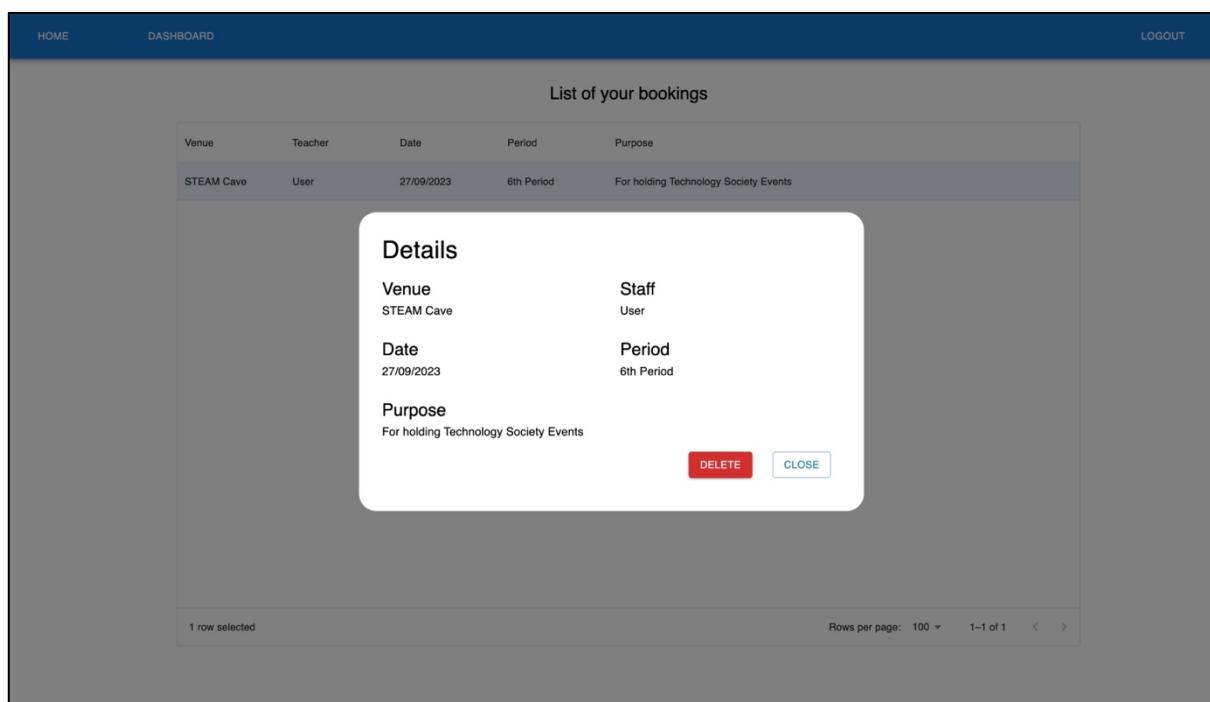


The screenshot shows a web application interface with a blue header bar containing 'HOME', 'DASHBOARD', and 'LOGOUT'. The main content area is titled 'List of your bookings' and contains a table with the following data:

Venue	Teacher	Date	Period	Purpose
STEAM Cave	User	27/09/2023	6th Period	For holding Technology Society Events

At the bottom right of the table, there is a pagination control showing 'Rows per page: 100' and '1-1 of 1'.

You will be redirect to the home page of the system after login, on this page, it will display the current bookings you have placed. You can simply read the detail by single clicking the row of the related record and a modal will pop up.



The screenshot shows the same 'List of your bookings' dashboard, but with a 'Details' modal open. The modal displays the following information:

Details	
Venue	Staff
STEAM Cave	User
Date	Period
27/09/2023	6th Period
Purpose	
For holding Technology Society Events	

At the bottom right of the modal, there are two buttons: 'DELETE' (red) and 'CLOSE' (blue). The background table is dimmed, and the pagination control at the bottom shows '1 row selected'.

This modal allows you to see the details of the selected booking and you can cancel it by simply clicking the delete button. Refresh your page and the booking will be removed from this list.

Checking all bookings

Head over dashboard by clicking the button on the app bar.

HOME

DASHBOARD

LOGOUT

Overview

Calendar

Profile

Overview of all bookings

CREATE NEW BOOKING

Venue	Teacher	Date	Period	Purpose
STEAM Cave	Admin	29/09/2023	4th Period	Lesson Activity
GP Room	Admin	12/10/2023	Lunch	Lunch Meetings
STEAM Cave	User	27/09/2023	6th Period	For holding Technology Society Events
Inno Lab	User	23/09/2023	3rd Period	For something fun
Inno Lab	User	23/09/2023	5th Period	I have no idea what I am doing again
MMLC	User	23/09/2023	4th Period	Testing
1B	Test 2	01/01/1970	After School	Testing too
Hall	Test 3	01/01/1970	1st Period	Another system testing

Rows per page: 100 1-8 of 8

HOME

DASHBOARD

LOGOUT

Overview

Calendar

Profile

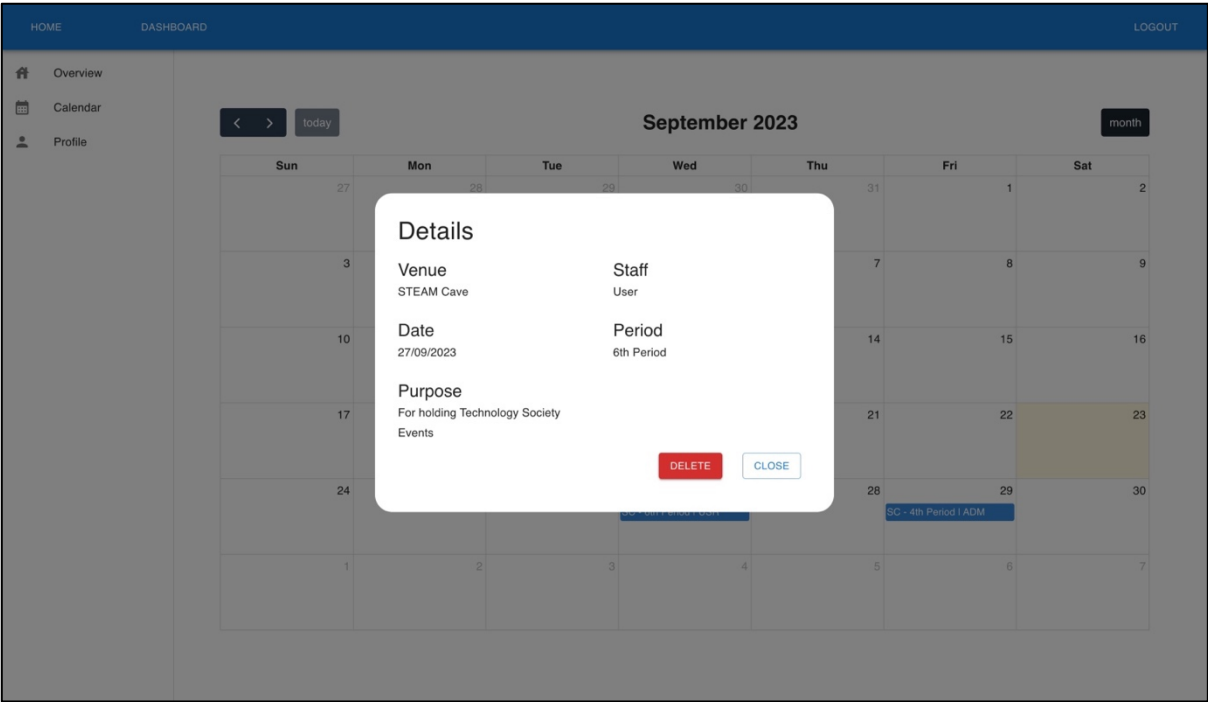
< > today

September 2023

month

Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
			1B - After School T2			
17	18	19	20	21	22	23
						I - 3rd Period USR I - 5th Period USR MC - 4th Period USR
24	25	26	27	28	29	30
			SC - 6th Period USR		SC - 4th Period ADM	
1	2	3	4	5	6	7

There provide two different views for all bookings, one is list and one is calendar.
You can also click on the booking, and it will pop up the modal of the booking details.



This modal just like the one on home page, you can also cancel your bookings through this modal. Refresh the page after deletion and the related booking will be removed.

Date ↑	
11/10/2023	
12/10/2023	
13/09/2023	
23/09/2023	
23/09/2023	
23/09/2023	
27/09/2023	
29/09/2023	

er	Date ↑		Pt
3		Sort by DESC	1s
n		Unsort	Lt
2		Filter	Af
		Hide column	3r
		Manage columns	5t
	23/09/2023		4t
	23/09/2023		6t
n	29/09/2023		4t

The bookings can be easily sorted according to the field you want.

Venue ▼	Teacher	Date ↑	Period ↑	Purpose
Inno Lab	User	23/09/2023	5th Period	I have no idea what I am doing again

Columns

Operator

Value

× Venue ▼ contains ▼ Inno Lab

You are able to filter the bookings by applying filtering rules on the table.

Creating a New Booking

Go back to overview tab on dashboard, click the “create new booking” button located on the top right corner of the list.

Create New Booking

Venue *

Date 09/23/2023

Period *

Reason *

CANCEL **SUBMIT**

This is the booking modal. You can fill in the information about venue, date, period and the reason or purpose you are booking the room for.

Create New Booking

Venue *

Date 09/23/2023

Period *

Reason *

CANCEL **SUBMIT**

- Inno Lab
- STEAM Cave
- Idea Lounge
- MMLC
- GP Room
- Art Room
- Music Room
- Lecture Theatre
- IS Lab
- Physics Lab
- Chemistry Lab
- Biology Lab
- Hall
- DT Room
- Fitness Room
- 1A
- 1B
- 1C
- 1D
- 2A
- 2B
- 2C
- 2D
- 3A

You can choose either the classroom or special room.

HOME DASHBOARD LOGOUT

Overview
Calendar
Profile

Overview of all bookings

CREATE NEW BOOKING

Venue	Teacher
STEAM Cave	Admin
GP Room	Admin
STEAM Cave	User
Inno Lab	User
Inno Lab	User
MMLC	User

Create New Booking

Venue *

Date

09/23/2023

October 2023

S M T W T F S

1 2 3 4 5 6 7

8 9 10 11 12 13 14

15 16 17 18 19 20 21

22 23 24 25 26 27 28

29 30 31

CANCEL SUBMIT

Rows per page: 100 1-6 of 6

Pick the date of your booking, Sunday is disabled as school is closed on Sunday.

HOME DASHBOARD LOGOUT

Overview
Calendar
Profile

Overview of all bookings

CREATE NEW BOOKING

Venue	Teacher
STEAM Cave	Admin
GP Room	Admin
STEAM Cave	User
Inno Lab	User
Inno Lab	User
MMLC	User

Create New Booking

Venue *

Date

09/23/2023

Period *

None

1st Period

2nd Period

3rd Period

4th Period

5th Period

6th Period

Lunch

7th Period

8th Period

9th Period

After School

CANCEL SUBMIT

Rows per page: 100 1-6 of 6

Choose the period afterwards.

Remember to fill in the reason for booking the venue.
Then press the submit button to confirm your booking.

Checking and Update Your Information

HOME

DASHBOARD

LOGOUT

Overview

Calendar

Profile

Profiles

Name

User

Email

user@dlthk.games

Password

user

UPDATE

Staff Initial

USR

This is Profile section of the dashboard, you can read your username, email, password and staff initial. You are able to update the password of your account on this page by typing the new password on the third field, click the update button to update your password.

localhost:3000 says

Password update failed!

OK

User

user@dlthk.games

UPDATE

USR

Any submission without input will return error to prevent you from updating the password to empty.

Conclusion

We have built a small web server for venue booking. It is completed with lots of function and a clear GUI. The system is easy to use and fulfilled most of the requirements of the objective.

However, there are still rooms for improvements. Most of the API and data fetching should be redesigned to minimize the traffic between frontend and backend of the server. We should force to use more secure password to increase the security of the system. Session token should be implemented to replace current session with user details like JWT (JSON Web Token) which will expired after a period to prevent session stolen by others. And admin panel is required to gain the top-level control of the whole system, including user and venue management.

I hope my further skill development can allow me to become more familiar with these development and implement more function within the same length of development.

Reference

Public

The documentation of the below websites is the reference of my project.

I used Material UI as the default theme for most of the components of the web design along with some customization CSS modification.

I use stackoverflow for searching the solution of the unknow and unresolvable problem faced during development.

React	https://react.dev/
Next.js	https://nextjs.org/
Material UI	https://mui.com/
Prisma	https://www.prisma.io/
Dayjs	https://day.js.org/
FullCalendar	https://fullcalendar.io/
StackOverflow	https://stackoverflow.com/

Private

All my push events and display of the site (If it is still hosting on my site and my domain hasn't change) can be check on below links.

GitHub	https://github.com/2LockTsun/Venue-Booking-System
Site Hosting	https://ictsba.dlthk.games/

Appendix

@=src/

@/prisma/schema.prisma

```
// This is the Prisma schema file

generator client {
  provider = "prisma-client-js"
}

// We are now using SQLite as our database
datasource db {
  provider = "sqlite"
  url      = "file:./dev.db"
}

// This is the model of Bookings in the database
model Booking {
  id Int @id @default(autoincrement())

  roomCode      String
  staffInitial   String
  purpose        String

  bookingPeriod String
  bookingDate    DateTime
  createDate     DateTime @default(now())
  updateDate     DateTime @updatedAt

  Staff Staff? @relation(fields: [staffInitial], references: [staffInitial])
  Venue Venue? @relation(fields: [roomCode], references: [roomCode])
}

// This is the model of Staff in the database
model Staff {
  id      Int      @id @default(autoincrement())
  admin   Boolean @default(false)

  name      String?
  email     String  @unique
  password   String
  staffInitial String @unique

  Booking Booking[]
}

// This is the model of Venues in the database
model Venue {
  roomCode String @id
  roomName String

  Booking Booking[]
}
```

@/prisma/seed.js

```
const { PrismaClient } = require('@prisma/client')
const prisma = new PrismaClient()

// This script is used to seed the database with initial data

async function main() {

  await prisma.venue.upsert({
    where: {roomCode: 'CH'},
    update: {},
    create: {roomCode: 'CH', roomName: 'Cyber Hub'},
  })
  await prisma.venue.upsert({
    where: {roomCode: 'I'},
    update: {},
    create: {roomCode: 'I', roomName: 'Inno Lab'},
  })
  await prisma.venue.upsert({
    where: {roomCode: 'SC'},
    update: {},
    create: {roomCode: 'SC', roomName: 'STEAM Cave'},
  })
  await prisma.venue.upsert({
    where: {roomCode: 'IL'},
    update: {},
    create: {roomCode: 'IL', roomName: 'Idea Lounge'},
  })
  await prisma.venue.upsert({
    where: {roomCode: 'MC'},
    update: {},
    create: {roomCode: 'MC', roomName: 'MMLC'},
  })
  await prisma.venue.upsert({
    where: {roomCode: 'GP'},
    update: {},
    create: {roomCode: 'GP', roomName: 'GP Room'},
  })
  await prisma.venue.upsert({
    where: {roomCode: 'A'},
    update: {},
    create: {roomCode: 'A', roomName: 'Art Room'},
  })
  await prisma.venue.upsert({
    where: {roomCode: 'M'},
    update: {},
    create: {roomCode: 'M', roomName: 'Music Room'},
  })
  await prisma.venue.upsert({
    where: {roomCode: 'LT'},
    update: {},
```

```

        create: {roomCode: 'LT', roomName: 'Lecture Theatre'},
    })
    await prisma.venue.upsert({
        where: {roomCode: 'IS'},
        update: {},
        create: {roomCode: 'IS', roomName: 'IS Lab'},
    })
    await prisma.venue.upsert({
        where: {roomCode: 'P'},
        update: {},
        create: {roomCode: 'P', roomName: 'Physics Lab'},
    })
    await prisma.venue.upsert({
        where: {roomCode: 'C'},
        update: {},
        create: {roomCode: 'C', roomName: 'Chemistry Lab'},
    })
    await prisma.venue.upsert({
        where: {roomCode: 'B'},
        update: {},
        create: {roomCode: 'B', roomName: 'Biology Lab'},
    })
    await prisma.venue.upsert({
        where: {roomCode: 'H'},
        update: {},
        create: {roomCode: 'H', roomName: 'Hall'},
    })
    await prisma.venue.upsert({
        where: {roomCode: 'DT'},
        update: {},
        create: {roomCode: 'DT', roomName: 'DT Room'},
    })
    await prisma.venue.upsert({
        where: {roomCode: 'FT'},
        update: {},
        create: {roomCode: 'FT', roomName: 'Fitness Room'},
    })
    await prisma.venue.upsert({
        where: {roomCode: '1A'},
        update: {},
        create: {roomCode: '1A', roomName: '1A'},
    })
    await prisma.venue.upsert({
        where: {roomCode: '1B'},
        update: {},
        create: {roomCode: '1B', roomName: '1B'},
    })
    await prisma.venue.upsert({
        where: {roomCode: '1C'},
        update: {},
        create: {roomCode: '1C', roomName: '1C'},
    })

```

```
    })
    await prisma.venue.upsert({
      where: {roomCode: '1D'},
      update: {},
      create: {roomCode: '1D', roomName: '1D'},
    })
    await prisma.venue.upsert({
      where: {roomCode: '2A'},
      update: {},
      create: {roomCode: '2A', roomName: '2A'},
    })
    await prisma.venue.upsert({
      where: {roomCode: '2B'},
      update: {},
      create: {roomCode: '2B', roomName: '2B'},
    })
    await prisma.venue.upsert({
      where: {roomCode: '2C'},
      update: {},
      create: {roomCode: '2C', roomName: '2C'},
    })
    await prisma.venue.upsert({
      where: {roomCode: '2D'},
      update: {},
      create: {roomCode: '2D', roomName: '2D'},
    })
    await prisma.venue.upsert({
      where: {roomCode: '3A'},
      update: {},
      create: {roomCode: '3A', roomName: '3A'},
    })
    await prisma.venue.upsert({
      where: {roomCode: '3B'},
      update: {},
      create: {roomCode: '3B', roomName: '3B'},
    })
    await prisma.venue.upsert({
      where: {roomCode: '3C'},
      update: {},
      create: {roomCode: '3C', roomName: '3C'},
    })
    await prisma.venue.upsert({
      where: {roomCode: '3D'},
      update: {},
      create: {roomCode: '3D', roomName: '3D'},
    })
    await prisma.venue.upsert({
      where: {roomCode: '4A'},
      update: {},
      create: {roomCode: '4A', roomName: '4A'},
    })
  })
```



```
await prisma.venue.upsert({
  where: {roomCode: '4B'},
  update: {},
  create: {roomCode: '4B', roomName: '4B'},
})
await prisma.venue.upsert({
  where: {roomCode: '4C'},
  update: {},
  create: {roomCode: '4C', roomName: '4C'},
})
await prisma.venue.upsert({
  where: {roomCode: '4D'},
  update: {},
  create: {roomCode: '4D', roomName: '4D'},
})
await prisma.venue.upsert({
  where: {roomCode: '5A'},
  update: {},
  create: {roomCode: '5A', roomName: '5A'},
})
await prisma.venue.upsert({
  where: {roomCode: '5B'},
  update: {},
  create: {roomCode: '5B', roomName: '5B'},
})
await prisma.venue.upsert({
  where: {roomCode: '5C'},
  update: {},
  create: {roomCode: '5C', roomName: '5C'},
})
await prisma.venue.upsert({
  where: {roomCode: '5D'},
  update: {},
  create: {roomCode: '5D', roomName: '5D'},
})
await prisma.venue.upsert({
  where: {roomCode: '6A'},
  update: {},
  create: {roomCode: '6A', roomName: '6A'},
})
await prisma.venue.upsert({
  where: {roomCode: '6B'},
  update: {},
  create: {roomCode: '6B', roomName: '6B'},
})
await prisma.venue.upsert({
  where: {roomCode: '6C'},
  update: {},
  create: {roomCode: '6C', roomName: '6C'},
})
await prisma.venue.upsert({
```

```

    where: {roomCode: '6D'},},
    update: {},
    create: {roomCode: '6D', roomName: '6D'},
  })

```

```

await prisma.staff.upsert({
  where: { staffInitial: 'ADM' },
  update: {},
  create: {
    admin: true,
    name: 'Admin',
    email: 'admin@dlthk.games',
    password: 'admin',
    staffInitial: 'ADM',
  }
})

```

```

await prisma.staff.upsert({
  where: { staffInitial: 'USR' },
  update: {},
  create: {
    admin: false,
    name: 'User',
    email: 'user@dlthk.games',
    password: 'user',
    staffInitial: 'USR',
  }
})

```

```

await prisma.staff.upsert({
  where: { staffInitial: 'KA' },
  update: {},
  create: {
    admin: false,
    name: 'Chan Ka Lun',
    email: 'ka@cpu.edu.hk',
    password: 'ka',
    staffInitial: 'KA',
  }
})

```

```

await prisma.staff.upsert({
  where: { staffInitial: 'YA' },
  update: {},
  create: {
    admin: false,
    email: 'ya@cpu.edu.hk',
    password: 'ya',
    staffInitial: 'YA',
  }
})

```

```
await prisma.staff.upsert({
  where: { staffInitial: 'T1' },
  update: {},
  create: {
    admin: false,
    name: 'Test 1',
    email: 'test1@test.com',
    password: '123456',
    staffInitial: 'T1',
  }
})
await prisma.staff.upsert({
  where: { staffInitial: 'T2' },
  update: {},
  create: {
    admin: false,
    name: 'Test 2',
    email: 'test2@test.com',
    password: '123456',
    staffInitial: 'T2',
  }
})
await prisma.staff.upsert({
  where: { staffInitial: 'T3' },
  update: {},
  create: {
    admin: false,
    name: 'Test 3',
    email: 'test3@test.com',
    password: '123456',
    staffInitial: 'T3',
  }
})
await prisma.staff.upsert({
  where: { staffInitial: 'T4' },
  update: {},
  create: {
    admin: false,
    name: 'Test 4',
    email: 'test4@test.com',
    password: '123456',
    staffInitial: 'T4',
  }
})
await prisma.staff.upsert({
  where: { staffInitial: 'T5' },
  update: {},
  create: {
    admin: false,
    name: 'Test 5',
```

```

        email: 'test5@test.com',
        password: '123456',
        staffInitial: 'T5',
      }
    })
  }

main()
  .then(async () => await prisma.$disconnect())
  .catch(async (err) => {
    console.error(err)
    await prisma.$disconnect()
    process.exit(1)
  })

```

@/pages/api/login.js

```

import { PrismaClient } from '@prisma/client'
const prisma = new PrismaClient()

export default async function handler(req, res) {
  // If the request method is not POST, return 405 Method Not Allow
  if (req.method !== 'POST') return res.status(405).json({ message: 'Method Not Allow' })

  // If the request body is not email and password, return 400 Bad Request
  const { email, password } = req.body
  if (!email || !password) return res.status(400).json({ message: 'Bad Request' })

  // If the email and password is not match, return 401 Unauthorized
  const target = await prisma.staff.findUnique({ where: { email: email } })
  if (!target) return res.status(401).json({ message: 'Unauthorized' })
  if (target.password !== password) return res.status(401).json({ message: 'Unauthorized' })
  return res.status(200).json({ message: 'OK', user: target })
}

```

@/pages/api/staff.js

```

import { PrismaClient } from "@prisma/client"

const prisma = new PrismaClient()

// This API endpoint is used to get all staffs

export default async function handler(req, res) {
  // If the request method is not GET, return 405 Method Not Allow
  if (req.method !== 'GET' && req.method !== 'PUT') return
  res.status(405).json({ status: 405, message: 'Method Not Allow' })
}

```

```

    // If the request method is GET and an id is provided, return the staff with
    the id
    // If the request method is PUT and an id is provided, update the staff with
    the id
    const id = req.query.id
    if (id && req.method === 'GET') return res.status(200).json({ status: 200,
    staff: await prisma.staff.findUnique({ where: { id: Number(id) } }) })
    if (id && req.method === 'PUT') {
        const { password } = req.body
        if (!password) return res.status(400).json({ status: 400, message: 'Bad
    Request' })
        return res.status(200).json({ status: 200, staff: await
    prisma.staff.update({ where: { id: Number(id) }, data: { password } }) })
    }

    // If the request method is GET but no id is provided, return 403 Forbidden
    return res.status(403).json({ status: 403, message: 'Forbidden' })
}

```

@/pages/api/venue.js

```

import { PrismaClient } from "@prisma/client"

const prisma = new PrismaClient()

// This API endpoint is used to access all venues

export default async function handler(req, res) {

    // If the request method is not GET, return 405 Method Not Allow
    if (req.method !== 'GET') return res.status(405).json({ status: 405, message:
    'Method Not Allow' })

    // If the request method is GET, return all venues
    return res.status(200).json({ status: 200, venues: await
    prisma.venue.findMany() })
}

```

@/pages/api/booking.js

```

import { PrismaClient } from "@prisma/client"
import dayjs from 'dayjs'

const prisma = new PrismaClient()

const validDate = dayjs().hour(0).minute(0).second(0).millisecond(0).toDate()

export default async function handler(req, res) {
    // If the request method is not POST or GET, return 405 Method Not Allow

```

```

    if(req.method !== 'POST' && req.method !== 'GET') return
    res.status(405).json({ status: 405, message: 'Method Not Allow' })

    // If the request method is POST, check with different requirements and
    create a new booking
    if(req.method === 'POST') {
        const { purpose, bookingDate, bookingPeriod, roomCode, staffInitial } =
        req.body
        if(!purpose || !bookingDate || !bookingPeriod || !roomCode
        || !staffInitial) return res.status(400).json({ status: 400, message: 'Bad
        Request' })

        const count = await prisma.booking.count({where: {staffInitial,
        bookingDate: {gte: validDate}}})
        if(count >= 4) return res.status(403).json({ status: 403, message:
        'Forbidden' })

        const bookings = await prisma.booking.findMany({where: {bookingDate,
        bookingPeriod, roomCode}})
        if(bookings.length > 0) return res.status(409).json({ status: 409,
        message: 'Conflict' })

        return res.status(200).json({ status: 200, booking: await
        prisma.booking.create({ data: req.body }) })
    }

    // If the request method is GET, return all bookings or a booking with the id
    or all bookings of a staff with the id
    const bookingId = Number(req.query.bookingId)
    const userId = Number(req.query.userId)
    if(!bookingId && !userId) return res.status(200).json({status: 200, bookings:
    await prisma.booking.findMany({where: {}, include: {Staff: true, Venue: true}})})
    if(bookingId) return res.status(200).json({status: 200, booking: await
    prisma.booking.findUnique({where: {id: bookingId}, include: {Staff: true, Venue:
    true}})})
    if(userId) return res.status(200).json({status: 200, bookings: await
    prisma.booking.findMany({where: {Staff: {id: userId}}, include: {Staff: true,
    Venue: true}})})
}

```

@/pages/api/booking/[id].js

```
import { PrismaClient } from "@prisma/client"

const prisma = new PrismaClient()

export default async function handler(req, res) {
  // If the request method is not DELETE or PUT, return 405 Method Not Allow
  if(req.method !== 'DELETE' && req.method !== 'PUT') return
  res.status(405).json({ status: 405, message: 'Method Not Allow' })

  // If the request method is DELETE, delete the booking with the id
  if(req.method === 'DELETE') return res.status(200).json({status: 200, data:
  await prisma.booking.delete({where: {id: Number(req.query.id)}})})

  // If the request method is PUT, update the booking with the id
  return res.status(200).json({
    status: 200,
    message: await prisma.booking.update({
      where: {id: Number(req.query.id)},
      data: req.body
    })
  })
}
```

@/app/layout.jsx

```
import './globals.css';

// This is the root layout for the entire app
export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <head />
      <body>{children}</body>
    </html>
  );
}
```

@/app/head.jsx

```
// Head component for the Home page
export default function Head() {
  return (
    <>
      <title>Home | Venue Booking System</title>
      <meta content="width=device-width, initial-scale=1" name="viewport" />
      <meta name="description" content="Venue Booking System by 2_Lock_Tsun" />
      <link rel="icon" href="/favicon.ico" />
    </>
  );
}
```

@/app/page.jsx

```
'use client';
import { useEffect, useState } from 'react';
import { useRouter } from 'next/navigation';
import { Box, Grid, Toolbar, Typography } from '@mui/material';
import { DataGrid } from '@mui/x-data-grid';

import CustomAppBar from '@components/global/CustomAppBar';
import BookingInfoModal from '../components/global/BookingInfoModal';

async function fetchBookings(userId) {
  const bookings = await fetch(
    `http://localhost:3000/api/booking?userId=${userId}`
  )
    .then((res) => res.json())
    .then((data) =>
      data.bookings.map((booking) => {
        return {
          id: booking.id,
          venue: booking.Venue.roomName,
          staff: booking.Staff.name,
          date: new Date(booking.bookingDate).toLocaleDateString(),
          period: booking.bookingPeriod,
          purpose: booking.purpose
        };
      })
    );
  return bookings;
}

async function fetchBooking(bookingId) {
  const booking = await fetch(
    `http://localhost:3000/api/booking?bookingId=${bookingId}`
  )
    .then((res) => res.json())
    .then((data) => data.booking);
  return booking;
}

// Home page for displaying all bookings of the related staff with a list

export default function Main() {
  // Check if the user is logged in, and redirect it to login page if not
  const router = useRouter();
  useEffect(() => {
    const session = JSON.parse(localStorage.getItem('session'));
    if (!session) return router.push('/login');
    fetchBookings(session.id).then((bookings) => setBookings(bookings));
  }, []);
```



```

// State variables for this list as named
const [bookings, setBookings] = useState([]);
const [bookingInfoModalOpen, setBookingInfoModalOpen] = useState(false);
const [selectedBooking, setSelectedBooking] = useState();
const [selectionModel, setSelectionModel] = useState();

// useEffect to update the state variable when the selectionModel prop changes
useEffect(() => {
  if (selectionModel) {
    fetchBooking(selectionModel).then((booking) => {
      setSelectedBooking(booking);
    });
  }
}, [selectionModel]);

// The columns of the list
const columns = [
  { field: 'venue', headerName: 'Venue', width: 150 },
  { field: 'staff', headerName: 'Teacher', width: 150 },
  { field: 'date', headerName: 'Date', width: 150 },
  { field: 'period', headerName: 'Period', width: 150 },
  { field: 'purpose', headerName: 'Purpose', width: 300 }
];

// The actual layout of the page
return (
  <Box sx={{ display: 'flex', justifyContent: 'center' }}>
    <CustomAppBar />
    <Toolbar />
    <Box
      component="main"
      sx={{
        display: 'flex',
        height: '90vh',
        justifyContent: 'center',
        alignItems: 'center',
        flexDirection: 'column'
      }}
    >
      <Toolbar />
      <Grid container columns={12}>
        <Grid item xs={12}>
          <Typography
            component="h1"
            variant="h5"
            textAlign="center"
            sx={{ padding: '1em' }}
          >
            List of your bookings
          </Typography>
        </Grid>
      </Grid>
    </Box>
  </Box>
)

```

```

</Grid>
<BookingInfoModal
  open={bookingInfoModalOpen}
  onClose={() => setBookingInfoModalOpen(false)}
  booking={selectedBooking}
/>
<DataGrid
  onRowSelectionModelChange={(model) => {
    setSelectionModel(model[0]);
    setBookingInfoModalOpen(true);
  }}
  slots={{
    noRowsOverlay: () => (
      <Box
        display="flex"
        justifyContent="center"
        alignItems="center"
        height="100%"
      >
        <Typography variant="h6">You have no bookings!</Typography>
      </Box>
    )
  }}
  sx={{
    height: '45vh',
    width: '75vw'
  }}
  rows={bookings}
  columns={columns}
/>
</Box>
</Box>
);
}

```

@/app/login/head.jsx

```

// Head component for the Login page

export default function Head() {
  return (
    <>
      <title>Log in | Venue Booking System</title>
      <meta content="width=device-width, initial-scale=1" name="viewport" />
      <meta name="description" content="Venue Booking System by 2_Lock_Tsun" />
      <link rel="icon" href="/favicon.ico" />
    </>
  );
}

```

@/app/login/page.jsx

```
'use client';
import { useEffect, useState } from 'react';
import { useRouter } from 'next/navigation';
import {
  Box,
  TextField,
  Typography,
  Button,
  Container,
  CssBaseline
} from '@mui/material';

// Login page for logging in to the application

export default function Login() {
  // Clear the session when the page is loaded
  useEffect(() => {
    localStorage.setItem('session', null);
  }, []);

  // State variables for this page as named
  const router = useRouter();
  const [error, setError] = useState(false);
  const [reason, setReason] = useState('');

  // The actual page content
  return (
    <Container component="main" maxWidth="xs">
      <CssBaseline />
      <Box
        sx={{
          display: 'flex',
          flexDirection: 'column',
          justifyContent: 'center',
          height: '100vh'
        }}
      >
        <Typography paddingBottom="1em" component="h1" variant="h3">
          CPU Venue Booking System
        </Typography>
        <Typography component="h1" variant="h5">
          Log in
        </Typography>
        <Box
          component="form"
          noValidate
          onSubmit={(e) => {
            setError(false);
            setReason('');
            e.preventDefault();
          }}
        >

```

```

const data = new FormData(e.currentTarget);

// Send a POST request to the api to log in
fetch('/api/login', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    email: data.get('email'),
    password: data.get('password')
  })
}).then((res) => {
  if (res.status === 200) {
    return res.json().then((data) => {
      localStorage.setItem('session', JSON.stringify(data.user));
      router.push('/');
    });
  } else if (res.status === 400) {
    setReason('Credentials cannot be empty!');
  } else if (res.status === 401) {
    setReason('Invalid credentials!');
  } else {
    setReason('Unknown error!');
  }
  setError(true);
});
})
>
<TextField
  margin="normal"
  id="email"
  label="email"
  name="email"
  required
  fullWidth
  autoFocus
  helperText={reason}
  error={error}
/>
<TextField
  margin="normal"
  required
  fullWidth
  helperText={reason}
  error={error}
  name="password"
  label="Password"
  type="password"
  id="password"
/>
<Button
  type="submit"

```

```

        fullWidth
        variant="contained"
        sx={{ mt: 3, mb: 2 }}
      >
        Log in
      </Button>
    </Box>
  </Box>
</Container>
);
}

```

@/app/dashboard/head.jsx

```

// Head component for the the Dashboard page

export default function Head() {
  return (
    <>
      <title>Dashboard | Venue Booking System</title>
      <meta content="width=device-width, initial-scale=1" name="viewport" />
      <meta name="description" content="Venue Booking System by 2_Lock_Tsun" />
      <link rel="icon" href="/favicon.ico" />
    </>
  );
}

```

@/app/dashboard/page.jsx

```

'use client';
import { useState, useEffect } from 'react';
import { useRouter } from 'next/navigation';
import {
  Box,
  Drawer,
  Toolbar,
  List,
  ListItem,
  ListItemButton,
  ListItemText,
  ListItemIcon,
  CssBaseline
} from '@mui/material';
import HouseIcon from '@mui/icons-material/House';
import CalendarMonthIcon from '@mui/icons-material/CalendarMonth';
import PersonIcon from '@mui/icons-material/Person';

import CustomAppBar from '@components/global/CustomAppBar';
import Overview from '@components/dashboard/Overview';

```

```

import Profile from '@components/dashboard/Profile';
import Calendar from '@components/dashboard/Calendar';

// Helper functions for the page to fetch all bookings
async function fetchBookings() {
  const bookings = await fetch(`http://localhost:3000/api/booking`)
    .then((res) => res.json())
    .then((data) =>
      data.bookings.map((booking) => {
        return {
          id: booking.id,
          venue: booking.Venue.roomName,
          staff: booking.Staff.name,
          date: new Date(booking.bookingDate).toLocaleDateString(),
          period: booking.bookingPeriod,
          purpose: booking.purpose
        };
      })
    );
  return bookings;
}

// Helper functions for the page to fetch all venues
async function fetchVenues() {
  const venues = await fetch(`http://localhost:3000/api/venue`)
    .then((res) => res.json())
    .then((data) => data.venues);
  return venues;
}

// Dashboard page for displaying all bookings of the related staff with a list
// and a calendar
// Also contains a profile page for the staff to view and edit their profile

export default function Dashboard() {
  // State variables for this page as named
  const router = useRouter();
  const [session, setSession] = useState();
  const [screen, setScreen] = useState(0);
  const [bookings, setBookings] = useState([]);
  const [venues, setVenues] = useState([]);

  // Check if the user is logged in, and redirect it to login page if not
  useEffect(() => {
    const user = JSON.parse(localStorage.getItem('session'));
    if (!user) router.push('/login');
    setSession(user);

    // Fetch all bookings and venues from the database
    fetchBookings().then((bookings) => setBookings(bookings));
    fetchVenues().then((venues) => setVenues(venues));
  });
}

```

```

}, []);

return (
  <Box sx={{ display: 'flex' }}>
    <CssBaseline />
    <CustomAppBar />
    <Drawer
      variant="permanent"
      sx={{
        width: '15em',
        flexShrink: 0,
        [`& .MuiDrawer-paper`]: {
          width: '15em',
          boxSizing: 'border-box'
        }
      }}
    >
      <Toolbar />
      <List>
        {[
          { text: 'Overview', icon: <HouseIcon /> },
          { text: 'Calendar', icon: <CalendarMonthIcon /> },
          { text: 'Profile', icon: <PersonIcon /> }
        ]
        .map((item, index) => (
          <ListItem key={item.text} disablePadding>
            <ListItemButton onClick={() => setScreen(index)}>
              <ListItemIcon>{item.icon}</ListItemIcon>
              <ListItemText primary={item.text} />
            </ListItemButton>
          </ListItem>
        ))}
      </List>
    </Drawer>
    <Toolbar />
    <Box
      component="main"
      sx={{
        flexGrow: 1,
        height: '100vh',
        flexDirection: 'column',
        padding: '5em',
        marginLeft: '-4em'
      }}
    >
      <Toolbar />
      {
        [
          <Overview
            bookings={bookings}
            venues={venues}
            staffInitial={session?.staffInitial}

```

```

        />,
        <Calendar bookings={bookings} venues={venues} />,
        <Profile staff={session} />
      ][[screen]
    }
  </Box>
</Box>
);
}

```

@/components/dashboard/Calendar.jsx

```

'use client';
import { useEffect, useState } from 'react';
import { Box } from '@mui/material';
import dayjs from 'dayjs';
import FullCalendar from '@fullcalendar/react';
import dayGridPlugin from '@fullcalendar/daygrid';
import interactionPlugin from '@fullcalendar/interaction';
import timeGridPlugin from '@fullcalendar/timegrid';
import listPlugin from '@fullcalendar/list';

import BookingInfoModal from '../global/BookingInfoModal';

// Helper functions for fetching bookings from the database
async function fetchBookings() {
  const bookings = await fetch(`http://localhost:3000/api/booking`)
    .then((res) => res.json())
    .then((data) => data.bookings);
  return bookings;
}

// Helper functions for fetching booking by id
async function fetchBooking(bookingId) {
  const booking = await fetch(
    `http://localhost:3000/api/booking?bookingId=${bookingId}`
  )
    .then((res) => res.json())
    .then((data) => data.booking);
  return booking;
}

// Calendar component for displaying all bookings with a calendar

export default function Calendar() {
  // State variables for the calendar as named
  const [bookings, setBookings] = useState([]);
  const [bookingInfoModalOpen, setBookingInfoModalOpen] = useState(false);
  const [selectedBooking, setSelectedBooking] = useState();
  const [selectionModel, setSelectionModel] = useState();

```



```

// useEffect to update the state variable when the selectionModel prop changes
useEffect(() => {
  if (selectionModel) {
    fetchBooking(selectionModel).then((booking) => {
      setSelectedBooking(booking);
    });
  }
}, [selectionModel]);

// useEffect to update the state variable as the page loaded
useEffect(() => {
  fetchBookings().then((bookings) => setBookings(bookings));
}, []);

// Helper function to convert bookings fetched from api to events on calendar
const events = bookings.map((booking) => {
  return {
    title: `${booking.roomCode} - ${booking.bookingPeriod} |
${booking.staffInitial}`,
    date: dayjs(booking.bookingDate).format('YYYY-MM-DD'),
    allDay: true,
    id: booking.id
  };
});

// The actual layout of the page
return (
  <Box flex="1 1 100%">
    <BookingInfoModal
      open={bookingInfoModalOpen}
      onClose={() => setBookingInfoModalOpen(false)}
      booking={selectedBooking}
    />
    <FullCalendar
      height="75vh"
      plugins={[dayGridPlugin, interactionPlugin, listPlugin, timeGridPlugin]}
      initialView="dayGridMonth"
      headerToolbar={{
        left: 'prev,next today',
        center: 'title',
        right: 'dayGridMonth'
      }}
      events={events}
      eventClick={(info) => {
        setSelectionModel(info.event.id);
        setBookingInfoModalOpen(true);
      }}
    />
  </Box>
);
}

```

@/components/dashboard/Overview.jsx

```
'use client';
import { useEffect, useState } from 'react';
import { Box, Button, Typography } from '@mui/material';
import { DataGrid } from '@mui/x-data-grid';
import CreateBookingModal from '../global/CreateBookingModal';
import BookingInfoModal from '../global/BookingInfoModal';

// Helper functions for the DataGrid to fetch booking by id
async function fetchBooking(bookingId) {
  const booking = await fetch(
    `http://localhost:3000/api/booking?bookingId=${bookingId}`
  )
    .then((res) => res.json())
    .then((data) => data.booking);
  return booking;
}

// Overview component for displaying all bookings with a list
export default function Overview({ bookings, venues, staffInitial }) {
  // State variables for this list as named
  const [createBookingModalOpen, setCreateBookingModalOpen] = useState(false);
  const [bookingInfoModalOpen, setBookingInfoModalOpen] = useState(false);
  const [selectedBooking, setSelectedBooking] = useState();
  const [selectionModel, setSelectionModel] = useState();

  // useEffect to update the state variable when the selectionModel prop changes
  useEffect(() => {
    if (selectionModel) {
      fetchBooking(selectionModel).then((booking) => {
        setSelectedBooking(booking);
      });
    }
  }, [selectionModel]);

  // The columns of the list
  const columns = [
    { field: 'venue', headerName: 'Venue', width: 150 },
    { field: 'staff', headerName: 'Teacher', width: 150 },
    { field: 'date', headerName: 'Date', width: 150 },
    { field: 'period', headerName: 'Period', width: 150 },
    { field: 'purpose', headerName: 'Purpose', width: 300 }
  ];

  // The actual layout of the page
  return (
    <Box flex="1 1 100%" sx={{ flexDirection: 'column' }}>
      <Typography
        component="h1"
        variant="h4"
      />
    </Box>
  );
}
```

```

        sx={{ padding: '1rem', marginTop: '-1.5em' }}
      >
        Overview of all bookings
      </Typography>
      <Box
        display="flex"
        sx={{ flexDirection: 'row', justifyContent: 'flex-end' }}
      >
        <Button
          variant="outlined"
          sx={{ margin: '1rem' }}
          onClick={() => setCreateBookingModalOpen(true)}
        >
          Create New Booking
        </Button>
      </Box>
      <BookingInfoModal
        open={bookingInfoModalOpen}
        onClose={() => setBookingInfoModalOpen(false)}
        booking={selectedBooking}
      />
      <CreateBookingModal
        open={createBookingModalOpen}
        onClose={() => setCreateBookingModalOpen(false)}
        staffInitial={staffInitial}
        venues={venues}
      />
      <DataGrid
        onRowSelectionModelChange={(model) => {
          setSelectionModel(model[0]);
          setBookingInfoModalOpen(true);
        }}
        slots={{
          noRowsOverlay: () => (
            <Box
              display="flex"
              justifyContent="center"
              alignItems="center"
              height="100%"
            >
              <Typography variant="h6">You have no bookings!</Typography>
            </Box>
          )
        }}
        sx={{ height: '75vh' }}
        rows={bookings}
        columns={columns}
      />
    </Box>
  );
}

```

@/components/dashboard/Profile.jsx

```
'use client';
import { useState } from 'react';
import { Box, Button, Grid, TextField, Typography } from '@mui/material';

// Profile component for displaying staff profile

export default function Profile({ staff }) {
  const [password, setPassword] = useState(staff.password);
  return (
    <Box flex="1 1 100%">
      <Typography
        component="h1"
        variant="h4"
        sx={{ padding: '1rem', marginTop: '-1.5em' }}
      >
        Profiles
      </Typography>
      <Grid container spacing={3} columns={12}>
        <Grid item xs={4}>
          <Typography component="h2" variant="h5">
            Name
          </Typography>
        </Grid>
        <Grid item xs={8}>
          <Typography variant="subtitle1">{staff.name}</Typography>
        </Grid>
        <Grid item xs={4}>
          <Typography component="h2" variant="h5">
            Email
          </Typography>
        </Grid>
        <Grid item xs={8}>
          <Typography variant="subtitle1">{staff.email}</Typography>
        </Grid>
        <Grid item xs={4}>
          <Typography component="h2" variant="h5">
            Password
          </Typography>
        </Grid>
        <Grid item xs={4}>
          <TextField
            fullWidth
            id="password"
            variant="standard"
            value={password}
            onChange={(event) => {
              setPassword(event.target.value);
            }}
          />
        </Grid>
      </Grid>
    </Box>
  );
}
```

```

</Grid>
<Grid item xs={4}>
  <Button
    variant="contained"
    onClick={() => {
      // UPDATE the password of the staff on the database
      fetch(`/api/staff?id=${staff.id}`, {
        method: 'PUT',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({
          password: password
        })
      }).then((res) => {
        if (res.status === 200) {
          alert('Password updated!');
        } else {
          alert('Password update failed!');
        }
      });
    }}
  >
    Update
  </Button>
</Grid>
<Grid item xs={4}>
  <Typography component="h2" variant="h5">
    Staff Initial
  </Typography>
</Grid>
<Grid item xs={8}>
  <Typography variant="subtitle1">{staff.staffInitial}</Typography>
</Grid>
</Grid>
</Box>
);
}

```

@/components/global/BookingInfoModal.jsx

```

import { useEffect, useState } from 'react';
import { Modal, Box, Typography, Button, Grid } from '@mui/material';

// BookingInfoModal component for displaying booking details

export default function BookingInfoModal({ open, onClose, booking }) {
  // State variables and its initial state for the booking to display as named
  const [bookingInfo, setBookingInfo] = useState({
    id: 'N/A',

```

```

Venue: { roomName: 'N/A' },
Staff: { name: 'N/A' },
bookingDate: 'N/A',
bookingPeriod: 'N/A',
purpose: 'N/A'
});

// useEffect to update the state variable when the booking prop changes
useEffect(() => {
  if (!booking) return;
  setBookingInfo(booking);
}, [booking]);

// The actual layout of the modal
return (
  <Modal
    open={open}
    onClose={onClose}
    aria-labelledby="modal-modal-title"
    aria-describedby="modal-modal-description"
    sx={{ display: 'flex', alignItems: 'center', justifyContent: 'center' }}
  >
    <Box
      sx={{
        width: '40em',
        borderRadius: '1.5em',
        bgcolor: 'white',
        padding: '2em',
        flexDirection: 'column'
      }}
    >
      <Typography component="h1" variant="h4" marginBottom="0.5em">
        Details
      </Typography>
      <Grid container spacing={3}>
        <Grid item xs={12} md={6}>
          <Typography component="h2" variant="h5">
            Venue
          </Typography>
          <Typography component="h2" variant="subtitle1">
            {bookingInfo.Venue.roomName}
          </Typography>
        </Grid>
        <Grid item xs={12} md={6}>
          <Typography component="h2" variant="h5">
            Staff
          </Typography>
          <Typography component="h2" variant="subtitle1">
            {bookingInfo.Staff.name}
          </Typography>
        </Grid>
      </Grid>
    </Box>
  </Modal>
)

```

```

<Grid item xs={12} md={6}>
  <Typography component="h2" variant="h5">
    Date
  </Typography>
  <Typography component="h2" variant="subtitle1">
    {new Date(bookingInfo.bookingDate).toLocaleDateString()}
  </Typography>
</Grid>
<Grid item xs={12} md={6}>
  <Typography component="h2" variant="h5">
    Period
  </Typography>
  <Typography component="h2" variant="subtitle1">
    {bookingInfo.bookingPeriod}
  </Typography>
</Grid>
<Grid item xs={12} md={6}>
  <Typography component="h2" variant="h5">
    Purpose
  </Typography>
  <Typography component="h2" variant="subtitle1">
    {bookingInfo.purpose}
  </Typography>
</Grid>
</Grid>
<Box
  sx={{
    display: 'flex',
    flexDirection: 'row',
    justifyContent: 'flex-end'
  }}
>
  <Button
    variant="contained"
    color="error"
    sx={{ margin: '1em' }}
    onClick={() => {
      // Delete the booking from the database
      fetch(`http://localhost:3000/api/booking/${bookingInfo.id}`, {
        method: 'DELETE'
      }).then((res) => {
        if (res.status == 200) onClose();
        else alert('Error deleting booking');
      });
    }}
  >
    Delete
  </Button>
  <Button
    variant="outlined"
    color="primary"

```

```

        sx={{ margin: '1em' }}
        onClick={onClose}
      >
        Close
      </Button>
    </Box>
  </Box>
</Modal>
);
}

```

@/components/global/CreateBookingModal.jsx

```

import { useState } from 'react';
import dayjs from 'dayjs';
import {
  Modal,
  Box,
  Typography,
  Button,
  TextField,
  Select,
  MenuItem,
  InputLabel,
  FormControl
} from '@mui/material';

import { AdapterDayjs } from '@mui/x-date-pickers/AdapterDayjs';
import { LocalizationProvider } from '@mui/x-date-pickers/LocalizationProvider';
import { DatePicker } from '@mui/x-date-pickers/DatePicker';

// Helper functions for the DatePicker
const today = dayjs();
const oneMonthFromToday = today.add(1, 'month');
const isSunday = (date) => {
  return date.day() === 0;
};

// CreateBookingModal component for creating new bookings

export default function CreateBookingModal({
  open,
  onClose,
  staffInitial,
  venues
}) {
  // State variables for the form as named
  const [venue, setVenue] = useState('');
  const [period, setPeriod] = useState('');
  const [reason, setReason] = useState('');
  const [error, setError] = useState(false);

```



```

const [date, setDate] = useState(today);

// The actual layout of the modal
return (
  <Modal
    open={open}
    aria-labelledby="modal-modal-title"
    aria-describedby="modal-modal-description"
    sx={{ display: 'flex', alignItems: 'center', justifyContent: 'center' }}
  >
    <Box
      component="form"
      noValidate
      onSubmit={(event) => {
        setError(false);
        setReason('');
        event.preventDefault();
        const data = new FormData(event.currentTarget);
        const bookingDate = date.hour(0).minute(0).second(0).millisecond(0);

        fetch('/api/booking', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({
            staffInitial: staffInitial,
            roomCode: data.get('venue'),
            bookingDate: bookingDate.toDate(),
            bookingPeriod: data.get('period'),
            purpose: data.get('reason')
          })
        }).then((res) => {
          if (res.status === 200) {
            onClose();
          } else if (res.status === 400) {
            setReason('Input cannot be empty!');
          } else if (res.status === 403) {
            setReason(
              'You are not authorized to create more than 4 bookings!'
            );
          } else if (res.status === 409) {
            setReason('Booking already exists!');
          } else {
            setReason('Unknown error!');
          }
          setError(true);
        });
      }}
      sx={{
        width: '40em',
        borderRadius: '1.5em',
        bgcolor: 'white',

```

```

        padding: '2em',
        flexDirection: 'column'
      }}
    >
    <Typography component="h1" variant="h4" marginBottom="0.5em">
      Create New Booking
    </Typography>
    <FormControl fullWidth required margin="normal" error={error}>
      <InputLabel id="venue-label">Venue</InputLabel>
      <Select
        id="venue"
        name="venue"
        label="Venue"
        labelId="venue-label"
        value={venue}
        onChange={(event) => setVenue(event.target.value)}
      >
        <MenuItem value={``} >None</MenuItem>
        {
          // Dynamically display the venues
          venues.map((venue, index) => (
            <MenuItem key={`menuItem${index}`} value={venue.roomCode}>
              {venue.roomName}
            </MenuItem>
          ))
        }
      </Select>
    </FormControl>
    <LocalizationProvider dateAdapter={AdapterDayjs}>
      <DatePicker
        label="Date"
        sx={{ marginTop: '0.5em' }}
        minDate={today}
        maxDate={oneMonthFromToday}
        shouldDisableDate={isSunday}
        value={date}
        onChange={(newDate) => setDate(newDate)}
      />
    </LocalizationProvider>
    <FormControl fullWidth required margin="normal" error={error}>
      <InputLabel id="period-label">Period</InputLabel>
      <Select
        id="period"
        name="period"
        label="Period"
        labelId="period-label"
        value={period}
        onChange={(event) => setPeriod(event.target.value)}
      >
        <MenuItem value={``} >None</MenuItem>
        {[

```

```

        '1st Period',
        '2nd Period',
        '3rd Period',
        '4th Period',
        '5th Period',
        '6th Period',
        'Lunch',
        '7th Period',
        '8th Period',
        '9th Period',
        'After School'
      ].map((period) => (
        <MenuItem key={`menuItem${period}`} value={period}>
          {period}
        </MenuItem>
      ))
    </Select>
  </FormControl>
  <TextField
    fullWidth
    required
    variant="standard"
    margin="normal"
    id="reason"
    name="reason"
    label="Reason"
    helperText={reason}
    error={error}
  />
  <Box
    sx={{
      display: 'flex',
      flexDirection: 'row',
      justifyContent: 'flex-end'
    }}
  >
    <Button
      variant="contained"
      color="error"
      sx={{ margin: '1em' }}
      onClick={onClose}
    >
      Cancel
    </Button>
    <Button
      variant="outlined"
      color="primary"
      sx={{ margin: '1em' }}
      type="submit"
    >
      Submit

```

```

        </Button>
      </Box>
    </Box>
  </Modal>
);
}

```

@/components/global/CustomAppBar.jsx

```

import { AppBar, Toolbar, Button, Grid } from '@mui/material';

// Custom AppBar for the application

export default function CustomAppBar() {
  return (
    <AppBar
      sx={{
        zIndex: (theme) => theme.zIndex.drawer + 1,
        flexDirection: 'row'
      }}
    >
      <Toolbar sx={{ width: '100vw' }}>
        <Grid container columns={10}>
          <Grid item xs={1}>
            <Button href="/" variant="home">
              Home
            </Button>
          </Grid>
          <Grid item xs={1}>
            <Button href="/dashboard" variant="dashboard">
              Dashboard
            </Button>
          </Grid>
        </Grid>
        <Grid item xs={1}>
          <Button href="/login" onClick={() => {}} variant="logout">
            Logout
          </Button>
        </Grid>
      </Toolbar>
    </AppBar>
  );
}

```