SeTMaS - Design Document

5th April, 2023

SWE Team-33

Jatin Tarachandani - CS20BTECH11021
Prashanth Sriram S - CS20BTECH11039
Shambhu Kavir - CS20BTECH11045
Taha Adeel Mohammed - CS20BTECH11052

1. Apply a top-down approach and divide your project into modules. List all your final factored modules and their functionalities in a table.

2. Provide class diagrams of all the classes in your project and their relationships/associations. In case your project does not use an object-oriented programming language, you can provide interfaces (or signatures) for each function and show how they access different data structures.

3. Draw a sequence diagram/activity diagram for one or two main use cases of your project.

4. List down the design patterns you used in your code.

MODULE VIEW

Sr.No	Module	Functionality Implemented
1	Authorization	It accepts the login credentials of a user and uses their IITH Gmail ID cross referenced with a list of admins to figure out the credentials of the user. It also allows the user to logout when they choose to.
2	Homepage	The homepage is the central linking page for all the functionalities of the application. It has links to all the user and admin functionality screens.
3	Calendar	It displays the calendar of upcoming seminars, with each day being expandable to see the seminars scheduled for that day.
4	Request Booking Portal	It displays a form style view for the user to create a

		booking request for a seminar they wish to hold. It includes fields like mailing lists, location and time, description etc.
5	Past Booking View	This allows a user to show all of their previously submitted requests. This will include the status of these bookings, such as accepted, rejected or pending.
6	(Admin) Pending Requests	This allows an admin to see all users' submitted requests. They can then decide to approve or reject them based on availability.
7	(Admin) Accepted Requests	This allows an admin to see the list of accepted requests. An admin can decide to cancel an approved request from this screen.
8	(Admin) List of Admins	This allows an admin to see a list of the email IDs of all the admins of the application. In the case of a superadmin, this view will be modifiable with the superadmin able to add, remove, and transfer super privileges through this module.
9	(System) Mailer	This allows the system to interface with the Gmail API and send emails to admins and users based on actions like creating, approving and rejecting requests. It will also be configured to send reminders to seminars, based on the requester's preferences.

CLASS DIAGRAMS

SEQUENCE DIAGRAM EXAMPLES

DESIGN PATTERNS USED

1. Decorator

This pattern is used to add multiple layers of functionality over a single object, while still having its interface appear like that of the base object. For example, the homescreen of SeTMaS can be represented using decorator objects; depending on the user privileges, different

functionalities will be displayed. Only after the user's admin status has been confirmed will the code for viewing the admin list be wrapped around the base homescreen object.

2. Singleton

Singleton pattern is a pattern used for defining objects that should have only one instance of them active at all times. This is useful in the case of a database request handler, for example; having multiple instances of this object existing in memory could possibly result in conflicts when both of them attempt to write to the database at the same time. The request handler being singleton allows us to have more control when multiple clients are attempting to make requests at the same time.

3. MVC Architecture

The Model-View-Controller is a common architecture pattern used to design and create interfaces and the structure of an application, by dividing the code functionality into 3 parts: Model, to handle the application data and logic to manipulate it, View, to handle how to display the data to the user, and Controller to handle the user's inputs and appropriately update the Model and View. In our application, we use the MVC pattern to display our calendar, allowing us to easily handle the different views for the calendar, along with real time updates to the calendar data.

4. Adapter

Adapter is a pattern used for converting the interface of one object to that of another (the other one being an object whose interface is not as familiar or easy to use, such as a third party API. Here, our objects can have easily understandable methods to send requests to the database via the handler, and the handler can resolve all of the related processing and call the database API without the other classes needing to worry about it. It essentially defines a "new API" for the classes within the server code to use, and it translates the "new API calls" to the existing ones for simplicity.

5. Observer

Observer is a pattern used when objects need to be notified of modifications to the data of other objects. This pattern is also aptly called the publish-subscribe pattern, since the data that is changed is essentially published to the subscribed classes. Here, we do not have multiple subscribers, but since the calendar content needs to be updated every time the request table is changed, the calendar is notified.

6. Chain of Responsibility

This pattern is used when there are multiple successive operations in a chain-like dependency. Each handler in the chain can choose to process the request, or take it to the next handler in the chain. This is used for our 3 levels of authentication, since once a user is checked for validity, the system must check if they are an admin. If they are an admin, the system must check if they are a super admin. If these checks fail, that means that the auth level of the user is the one checked by the immediately previous handler, which is known to us through our structure.

Request

- title: string
- details: string
- venue: string
- startTime: date
- endTime: date
- department: string
- email: string
- mailingLists: string[]
- reminders: date[]

+ getState()

Request is basically a struct to hold the data pertaining to a request

AbstractCalendarView

+ Update()

Execute event

CalendarController

+ ControllerAlgorithm()

Update Model

Extends

Update View

CalendarMonthView

- monthViewState

+ Update()

CalendarDayView

- dayViewState

+ Update()

Subject

Subject

Calendar (Singleton)

- calendarState

+ Attach(in observer: AbstractCalendarView)
+ Detach(in observer: AbstractCalendarView)
+ NotifyView()
+ getState()
+ Update()

calendar->update()

BookingDatabaseAdapter (Singleton)

- calendar

+ notify()
+ getState()

subject

Request DB

return

dispatch

«interface»
Screen

+ render()

<<extend>>

Actor

IITHAuthHandler

+ verifyGoogleAuth(): bool

dispatch

AdminAuthHandler

+ verifyAdminAuth(): bool

dispatch

SuperAuthHandler

+ verifySuperAuth(): bool

control flow

Homescreen

+ render()

BaseDecorator

+ render()

ctrl flow (on failing next check)

on failing base check return to user

ctrl flow (on fail check)

Extends

Extends

UserDecorator

+ render()

AdminDecorator

+ render()

AdminDecorator is always wrapped over the UserDecorator

Lead to

Lead to

RequestBookingPage

+ render()

ViewBookingPage

+ render()

ViewPendingBookingPage

+ render()

AdminListPage

+ render()

Use

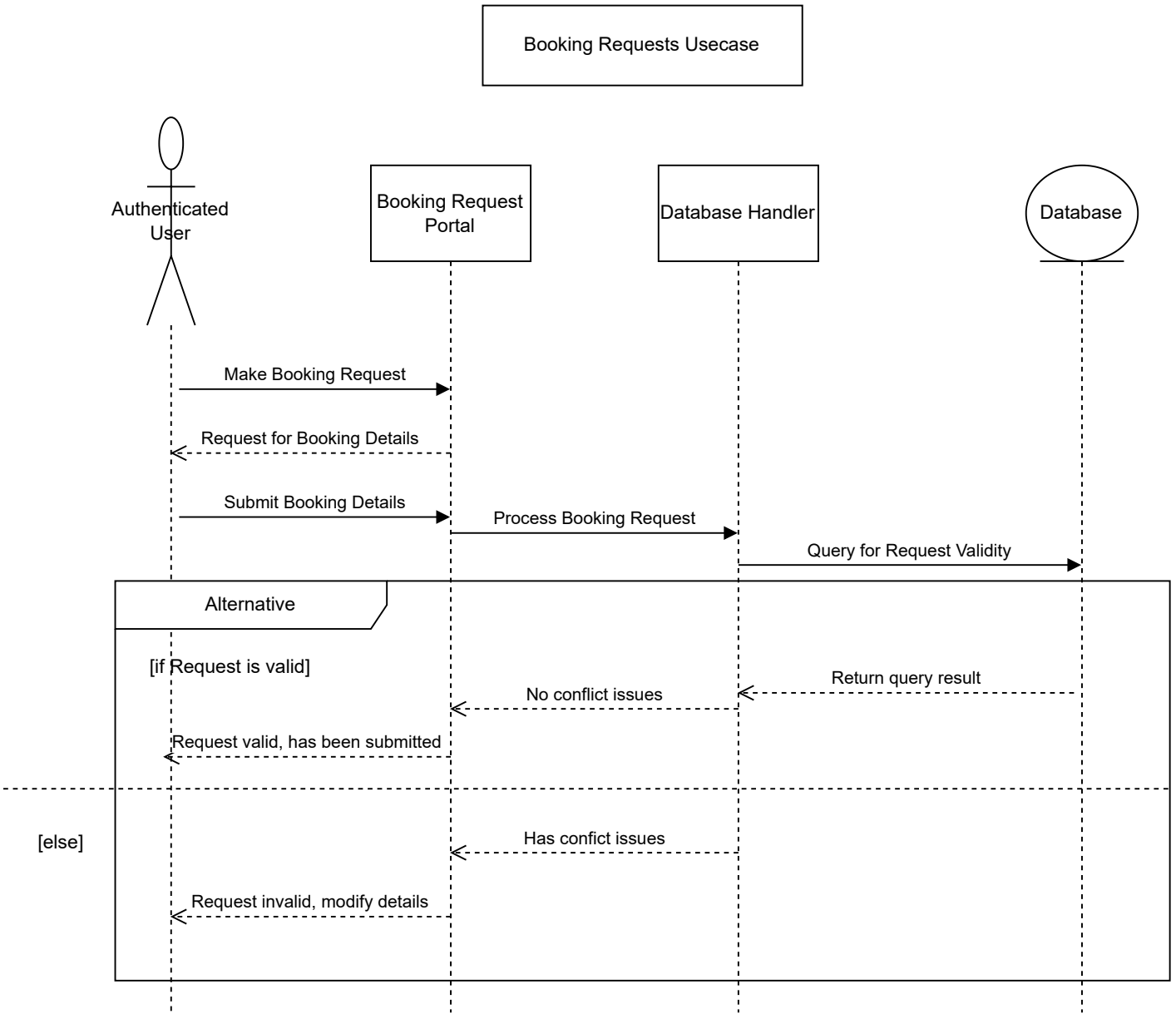
Use

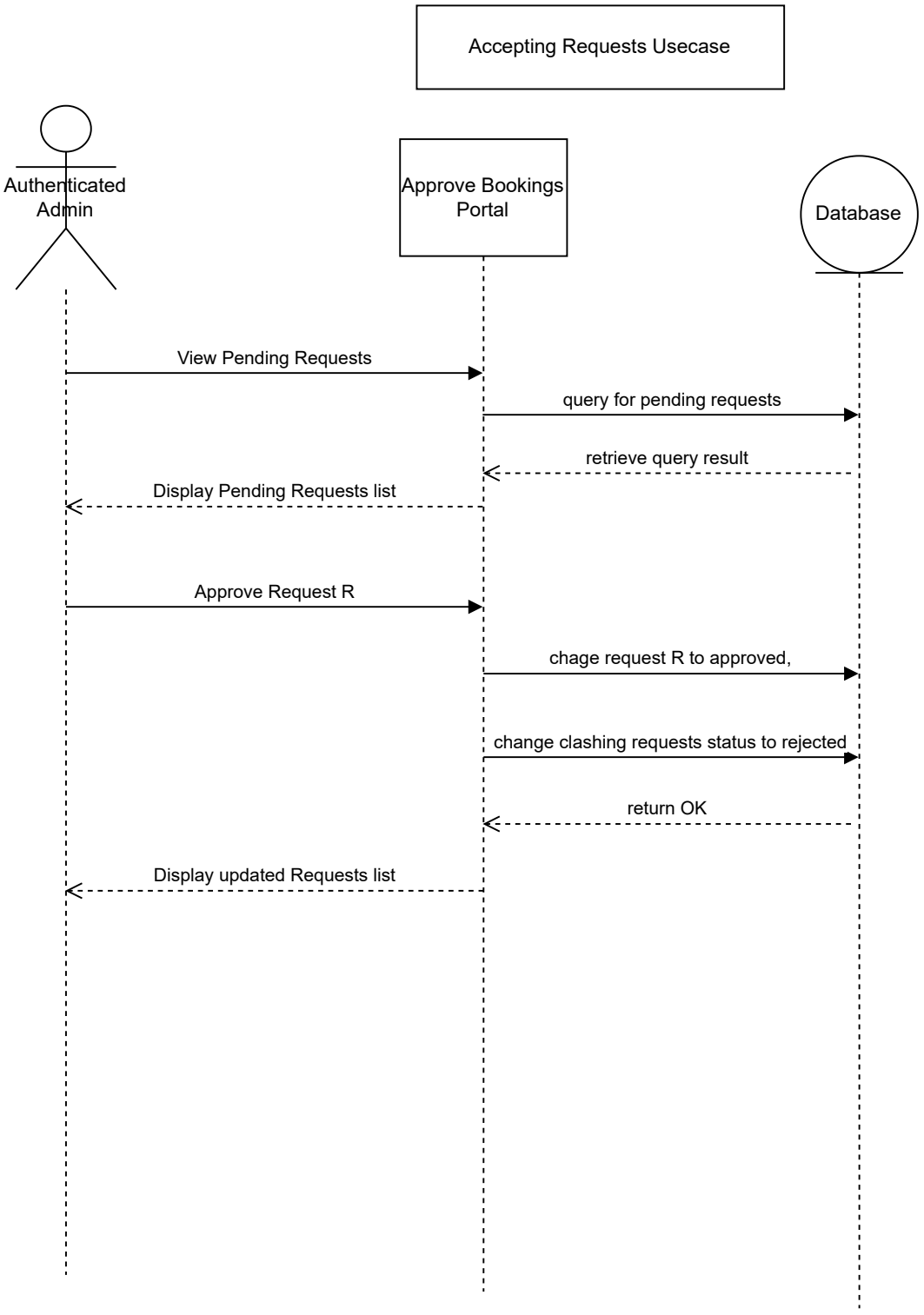
Use

Use

Use

All 4 objects pass requests to the DB handler, represented this way for clarity


Booking Requests Usecase
Authenticated User
Booking Request Portal
Database Handler
Database
Make Booking Request
Request for Booking Details
Submit Booking Details
Process Booking Request
Query for Request Validity
Alternative
[if Request is valid]
Return query result
No conflict issues
Request valid, has been submitted
[else]
Has conflict issues
Request invalid, modify details


Accepting Requests Usecase
Authenticated Admin
Approve Bookings Portal
Database
View Pending Requests
query for pending requests
retrieve query result
Display Pending Requests list
Approve Request R
chage request R to approved,
change clashing requests status to rejected
return OK
Display updated Requests list