1. Java and spring boot
2. User Stories
3. Wireframes
4. API contracts
5. Git and api security
6. Backend coding
7. Frontend-Js,React
8. Front end coding

—-----

1. API

What is an API?

What are restful APIs?

What are api contracts? (Resources)

API Security

1. Authentication
2. Authorization

JWT web tokens - working of jwt tokens (generation, validation)

1. Spring boot

Java framework

Features that helps in making application - Annotations, Dependency Injection, Spring Security, spring data JPA

1. React
2. Time line of my project

User stories

Designed wireframes

Worked what api would I be needing and how to write api contracts

A very pleasant morning to one and all present. Today, I would be presenting my project that I was doing as my part of OnGrid training.

The Payment portal application. Let's go through it.

Here in this presentation, we would be going through the following table of contents. To keep it brief and concise

Initially, I will be giving you the jist of my project. The project overview.

We would be going through the working timeline of the project and what were my major learnings that I gained throughout the working and implementation of the project.

And thereafter, I’ll show you the demo and then we will revolve our discussion around that.

So, what is this project? I tried making a payment portal application for the university. This application provides a convenience for the registered students in the university to pay their college fees. The prime features of this application and you see in the presentation as well. I tried to instill my application with features like a user friendly UI, user authentication, easy and fluent transaction management.

What timeline did I follow?

**User Stories**

First, I tried to capture the requirements the user would want to have from the application. I worked on describing the feature, or you can say the requirements from the perspective of the end user.

1. Now, students would be enrolled in various programs, like mtech and btech. So, students would want to select their respective programs to see the fee structure and what is the amount that he or she needs to pay.
2. Further, the user would want to log in to pay his fees.
3. After logging in, what is the obvious requirement? User pays his fees. Now, I thought of implementing a third party API for payment portals (like stripe, razorpay).
4. What else? So After making the payment, user would want to preview his receipt or invoice of the payment
5. Sounds good. Then, the user just wants to log out.

Adding to this, I also worked on writing a use case, the detailed description of how users would be interacting with the system.

**Wireframes**

Alright, then I made a wireframe for my application. The structure of the application would look like. I used the balsamiq software to do so.

**API**

Now, I am done with user stories. How would I implement these features? So here comes the picture of the API. Application programming Interface. We all know what that is, just for the record… API acts as interface between two software components so that they can communicate and interact with each other according to the to some rules and requirements (which call them as API contracts). So, I made API contracts for my application

What was my thought process in making the API. I'll explain to you with an example of user stories.

Lets take.. Umm. Okay, User would like to select his programme to see his fee structure. So, do I need an API for this? Can I just hardcode the values of fees in my frontend component? This was my initial thought. But, after some thinking and guidance from aditya. I thought what if in future I decide to change my fee structure. Then I would have to deploy my application again. We don't want that. So, it would be good. I fetch the fee structure of the programme via an API. that from my backend.

So, in order to create an API, things that we keep in mind are what would be the resource and maybe subresource, what would be my request body and what would be the response body.

So, I went back to my use stories.

So my user stories was - Users want to select respective programmes to get the fee structure and amount.

So from here I could see that a program ( a noun) is a resource. Now which programme. So I need to specify which program the user wants. So, I gained the knowledge of path variables.

Now, furthermore. What is the method that I have to use? I am just fetching my fee data. I am not updating and creating and instance in db. So it definitely would be Get Request.

Similarly, I made my other API.

The login api. I am authenticating the user. A post request in which a student is sending his credentials to authenticate himself. Why did I not use get requests here? Because the GET request doesn't contain a response body and we don't want our credentials to be displayed. I also gave special attention to frame api names so that all the api are self explanatory.

API - get programme, post login, post info, post payment, get receipt.

**DB Schemas**

Now comes the turn for designing my schemas

I need to have a database to store my record. I used a mysql database. What data do I want to store? What are the resources my APIs have?

Programme. User information. Payment information. So these are entities that I need. Then, I defined the attributes these entities would be requiring. For example, a user entity would basically have a username, rollNumber, password. Yeah that’s it. Okay, now I need to relate my entities. So I used the foreign keys and primary keys keeping in mind mapping cardinality, whether the entities are mapped one-to-one, one-to-many and so on. Before u actually create a relational model from entity relation diagram u need to perform a check. I followed the basic normal forms (1NF, 2NF). Like no attribute should be composite. For example, I destructured the user name into first name and last name.

So, working on these ideas on paper is one thing and implementing it/coding it is another thing.

Along with working on these ideas, I was doing my courses. Java and spring boot. With the knowledge I gathered from it, I tried to implement it.

So, in the end, I exploited the spring boot. So, here are key things that I tried to persist in my brain.

—-------------------

**JDBC, JPA, HIBERNATE**

JDBC (Java Database Connectivity): JDBC is a low-level API provided by Java for interacting with relational databases. Developers manually write SQL queries and handle database connections, statements, and result sets.

Hibernate:

Hibernate is a popular Object-Relational Mapping (ORM) framework that simplifies database access by mapping Java objects to database tables. Developers work with Java objects and let Hibernate handle the underlying SQL queries and database interactions. Hibernate is an implementation of the Java Persistence API (JPA).

pring Data JPA: Spring Data JPA is a specific module of Spring Data that builds on top of JPA (Java Persistence API). Provides repositories with pre-defined methods for common CRUD operations, reducing boilerplate code.

—------------------------------

**Authentication and Authorization**

Authentication and authorization are two fundamental concepts in the realm of security and access control, especially in the context of web applications and system.

Authentication: Authentication is the process of verifying the identity of a user, system, or entity.

*WHO ARE YOU?*

Authentication mechanisms include:

Username and Password: Users provide a username and password combination to prove their identity. This is the most common form of authentication.

Token-based Authentication: Tokens (like JSON Web Tokens or JWTs) are issued to users upon successful login. Tokens are then sent with subsequent requests to verify identity.

Authorization: Authorization is the process of determining whether a user, once authenticated, has the necessary permissions and privileges to access a particular resource or perform a specific action.

"*What are you allowed to do?"*

*Relationship between Authentication and Authorization: Authentication is the first step: verifying the identity of the user. Once authenticated, the system determines what the user is allowed to do based on their roles, permissions, and attributes. This is where authorization comes into play. An authenticated user can be authorized to perform certain actions or access certain resources, while being restricted from others.*

—----------------

JWT token to authenticate my user

—-----------

Routing

Routing in React refers to the process of managing different "routes" or URLs within a single-page application (SPA). It enables navigation between different views or components based on the URL, simulating the behavior of multiple pages in a traditional multi-page application. In a single-page application, all content is typically loaded into a single HTML page, and routing allows you to update the displayed content without actually navigating to a new HTML page. This provides a smoother and more interactive user experience. Key concepts and components related to routing in React include: Route: A route is a specific URL path mapped to a particular component. Each route defines which component should be rendered when the URL matches its path.