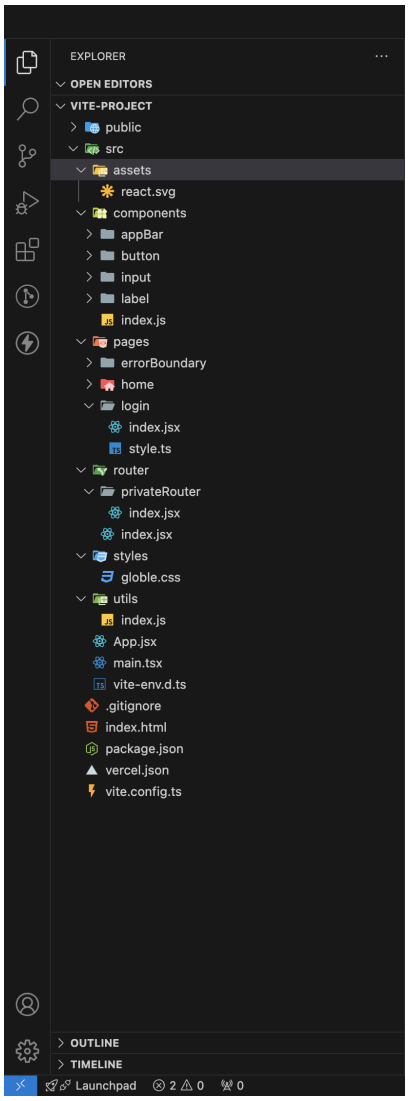
Fullstack - Crayon'd

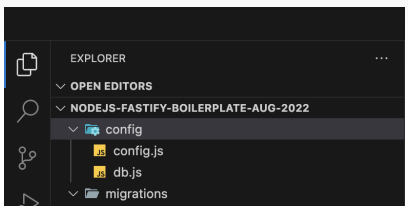
Day 1 - 19/09/2024:

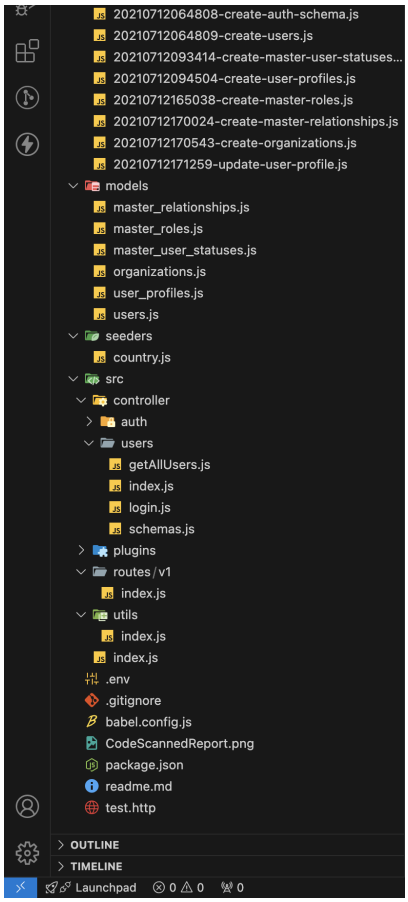
- Focused on setting up an **industry-level folder structure** for both **frontend** and **backend** development. This is crucial for ensuring **scalability** and **organization** in larger projects.

Front End boiler plate code



Backend End boiler plate code





- Introduced to **Sequelize**, an **Object-Relational Mapping (ORM)** library that simplifies interactions with relational databases in **Node.js**. Sequelize helps manage database models, migrations, and queries in a structured manner.

  Sequelize: An ORM for Node.js that provides a simple, clean syntax for database operations, making data manipulation easier and more efficient.

Getting Started with Sequelize

Step 1: Install Sequelize and CLI

- First, install Sequelize and Sequelize CLI along with the necessary database dependencies in your backend directory (e.g., PostgreSQL,MySQL,...):

```
npm install sequelize sequelize-cli
npm install mysql2  # For MySQL
```

Step 2: Initialize sequlize in backend directory

```
npx sequelize-cli init
```

- This command will generate the following directory structure:

```
├── config
│   └── config.json
├── models
│   └── index.js
├── migrations
└── seeders
```

Step 3: generate a model and migration folder

- Follow this step if you want to use class based syntax in model unless skip to **step 4**
- migration folder contains all the migration files which is use to create the schema of the database (structure)
- and the modal are represention od the table schema and used to intract with them

this command takes some flags for attributes like fullName , lastName, email

```
npx sequelize-cli model:generate --name User --attributes firstName:string,
lastName:string,email:string
```

this command will generate a sample user migration and model

Step 4: follow step 4 instead of step 3 if you prefer object based syntax for models

- for object based syntax for models we need to custom create the models thus we need to first generate migration files alone by using

```
npx sequelize-cli migration:generate --name migration_name
```

- Emphasized how this structured approach not only improves **collaboration** but also enhances **debugging** efficiency.
- Key takeaway: Understanding the **importance of separation of concerns** directly contributes to maintaining a **clean** and **scalable architecture**.

  This approach builds the foundation for long-term maintainability, which is critical in industry-level projects.

Day 2 - 20/09/2024:

- **Git** and **GitHub** were introduced for **version control** and **collaboration** in software development. We learned how to:
  - Initialize repositories

```
git init
```

- Create branches

```
git branch <branch_name>
```

- Switch between branches

```
git checkout <branch_name>
```

- Make pull requests (Pull requests are created on GitHub, typically through the GitHub interface after pushing branches.)
- Resolve merge conflicts (Merge conflicts are resolved during the merge process, typically by editing conflicting files and then committing the changes.)

  Git: A version control system that tracks changes in code, enabling multiple developers to collaborate without overwriting each other's work.

  GitHub: A web-based platform for hosting Git repositories, facilitating collaboration and code sharing among development teams.

- Continued working with **Sequelize**, exploring real-world applications to simplify database interactions. Hands-on practice was invaluable in understanding how Sequelize works in practical scenarios.
- Branching strategies were also covered, helping manage feature development in parallel without code conflicts.
- **Merge conflict resolution** was practiced, which is a critical skill in team-based development.

Common Git Commands used

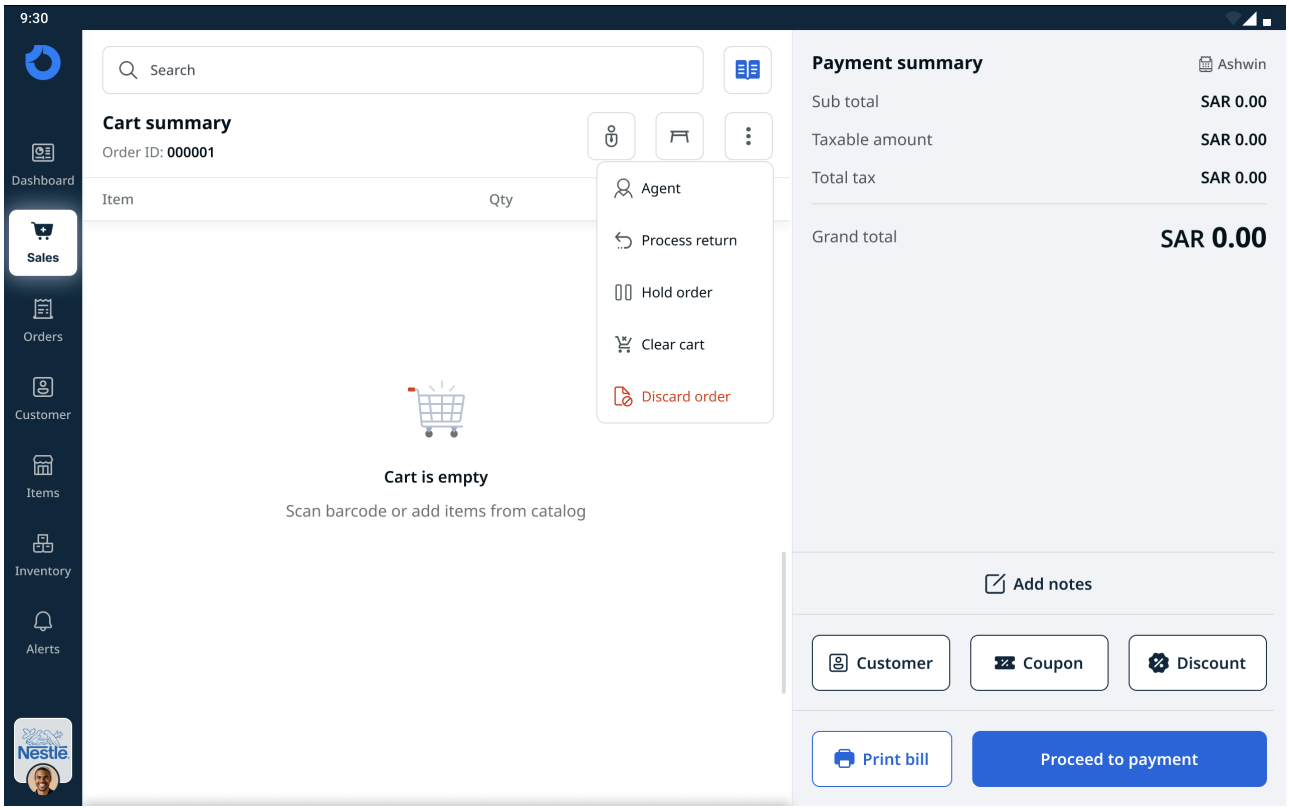
- **Clone a repository**

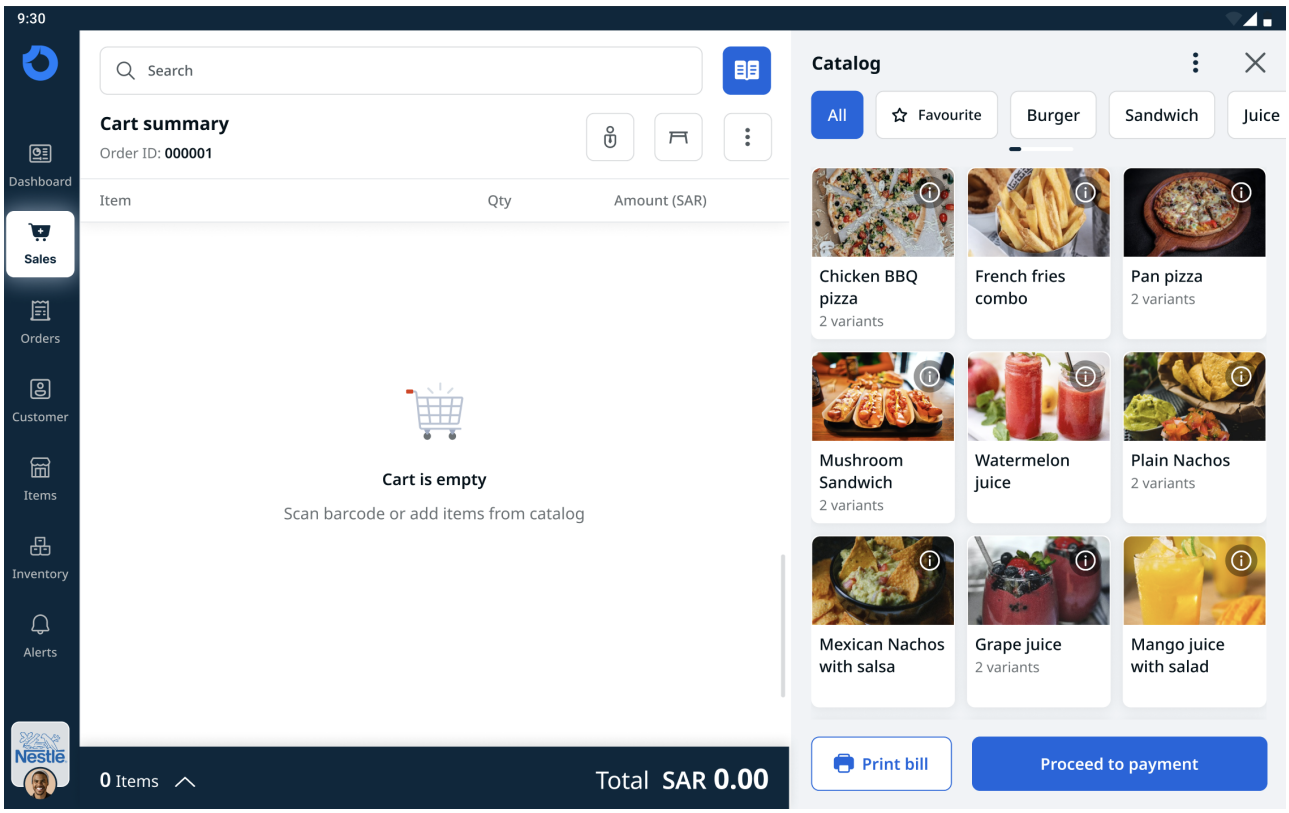
```
git clone <repository_url>

git status

git add <file_name>      # Add a specific file

git add .                # Add all changes

git commit -m "Commit message describing the changes" #Commit changes
```

```
git fetch origin # Pull the repository

git push origin <branch_name> # Push changes to a remote repository

git fetch origin #Fetch changes from a remote repository

git pull origin <branch_name> #Pull changes from a remote repository

git merge <branch_name> #Merge branches
```

Day 3 - 21/09/2024:

- A full-stack **eCommerce project** was assigned, which involved building both the frontend and backend using **Sequelize** for database management.





- **Folder structures** and **Git** were essential tools used throughout the project.
- The project covered everything from designing the **database models** to developing the **user interface**.
- This exercise encouraged teamwork and integration of different technologies, fostering better problem-solving skills.
- It gave us valuable experience in designing the architecture of a full application, from database models to the user interface.

- Overall, it was a great exercise in problem-solving and collaboration, preparing us for more complex, real-world scenarios.

  This project served as a bridge between theoretical knowledge and practical application, simulating real-world challenges faced in industry-level projects.

Day 4 - 23/09/2024:

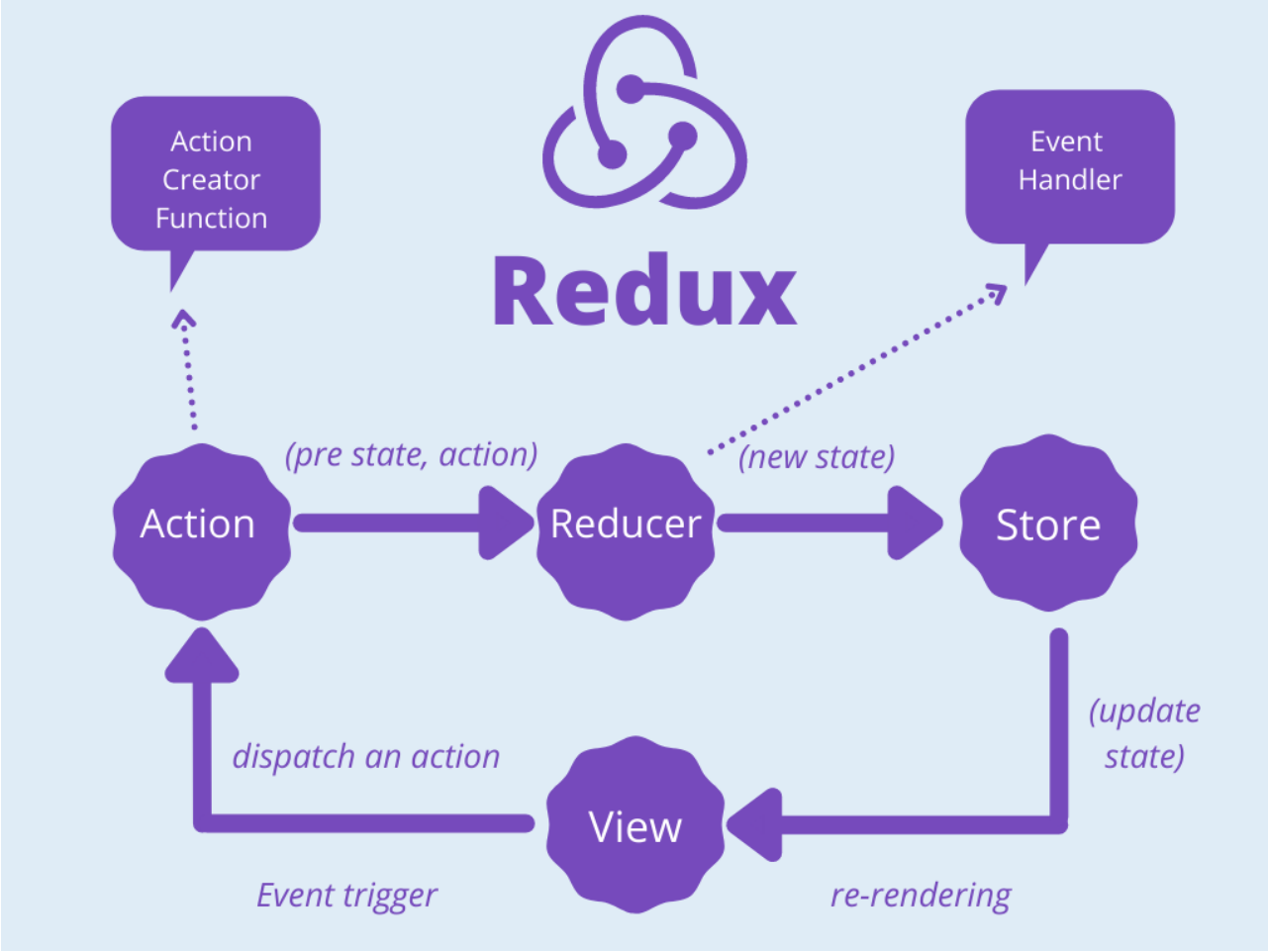
- The **Crayon team** reviewed our weekend project and provided detailed feedback. Areas of improvement included:
  - **Optimizing database queries**
  - **Refining UI components**
  - **Managing folder structure**
- Assigned smaller, focused tasks to refine these areas. This feedback-driven approach helped boost confidence in **problem-solving** and **debugging** real-world applications.

  Breaking down complex problems into manageable tasks is a key industry practice, ensuring that each component is thoroughly refined and optimized.

Day 5 - 24/09/2024:

- Advanced topics like **asynchronous programming** and **global state management** with **Redux** were introduced.

  Redux: A state management tool for JavaScript apps, providing a centralized place to manage application state and make data flow more predictable.



Redux helps manage complex data flows and user interactions across components.

- **React Hooks** and the **Context API** were also explored, allowing efficient data management without **prop drilling**.

  React Hooks: Functions that let you "hook into" React state and lifecycle features from functional components.

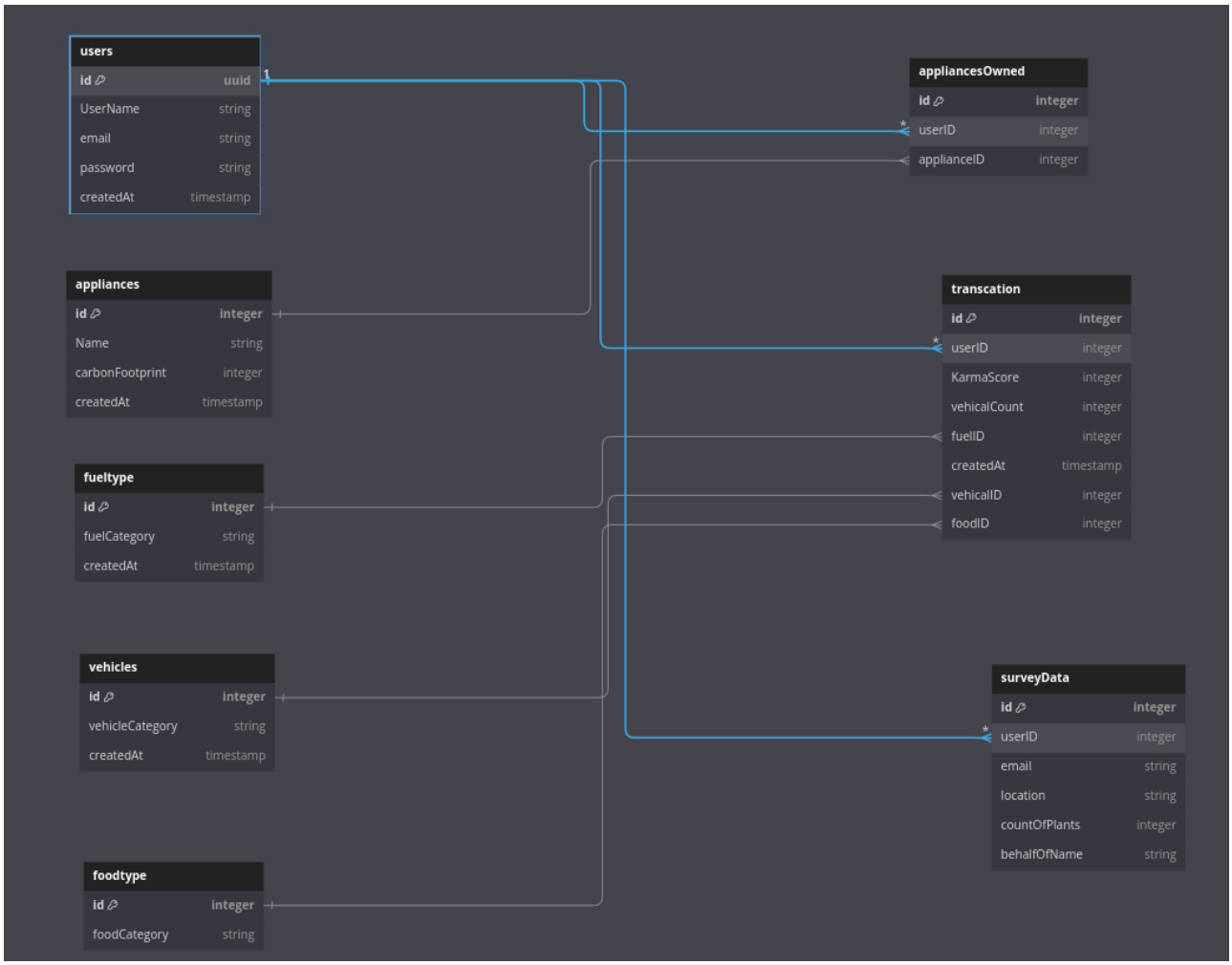
- Gained a better understanding of **scalable database architecture**.

  Learning how to manage both state and data efficiently is vital when working on dynamic, data-driven applications in a real-world setting.

Day 6 - 25/09/2024:

- Explored advanced Git techniques, including team collaboration and branch strategies.
- Hands-on practice with **Sequelize** and **Redux** further solidified our understanding of these concepts.
- Designed a **database structure** using the no-code platform **Db Diagram**. This helped visualize and plan out the architecture for scalable databases.

  Refining these skills is crucial for maintaining organized, efficient systems as the complexity of the project grows.



Day 7 - 26/09/2024:

- The seventh day focused on industry-level **code management** using the **Git flow** approach, a methodology for managing code across multiple teams and environments.

  Git Flow: A workflow design for Git that helps manage feature development and release cycles in production environments.

- Assigned to work on the **Karma Calculator** feature for an existing application, **Produkt**, as full-stack developers responsible for:

  - Backend logic and database management
  - Integrating the frontend with the backend

- This task gave us insights into managing **production-level code** and working in multi-developer environments.

  By applying Git Flow principles, we gained valuable experience in maintaining a robust, well-structured codebase, essential for any industry-level project.

- **UI Link**: Adobe UI