Technologies & Tools

Development Environment

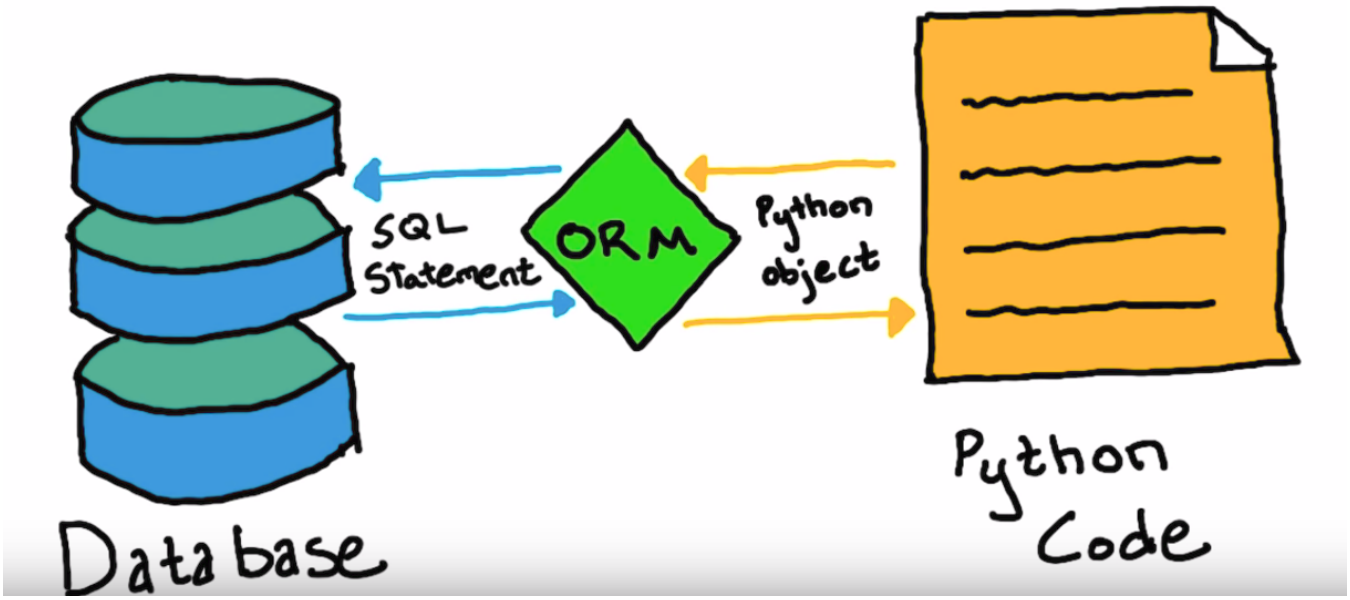
Before working on the project, we received the code of a team from the last semester. After one week of looking into the code, we realised that we **could not** reuse the code for some following reasons. First, the requirements of our project are totally different. Secondly, the code quality is very poor. Thirdly, the software design is truly terrible, especially on the frontend side. And finally, the technologies used are completely outdated. Consequently, we decided to make some BIG CHANGES for our project although we still kept some design ideas..

The project uses a micro-services architecture to arrange the web application as a set of loosely-coupled services, including a backend server, frontend client, and database. For more details, please read: Microservices Architecture

The specific technologies used for each of these microservices module are detailed below.

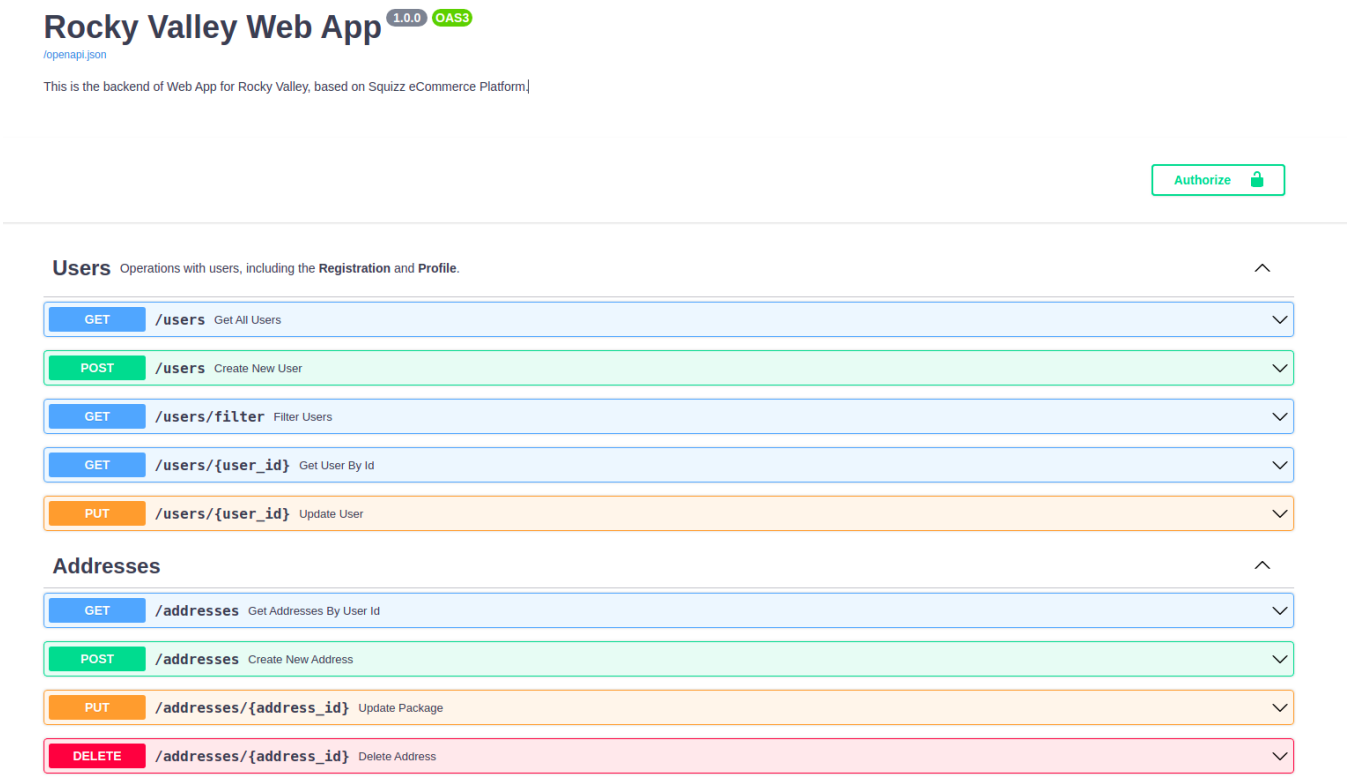
Backend

- Instead of using Flask (https://flask.palletsprojects.com/en/2.0.x/) like the last semester's team, we decided to select FastAPI (https://fastapi.tiangolo.com/) because we wanted to try something new and we all know that FastAPI is the state-of-the-art. In practice, FastAPI is well known to be the fastest python framework. It is **100 times** faster than Flask in any given situation (https://blog.accubits.com/flask-vs-fastapi-which-one-should-you-choose/)
- Instead of writing SQL queries to retrieve data from the database on the backend side, we adopted an ORM (Object Relational Mapping) framework named SQLAlchemy to get rid of writing complicated SQL queries.



ORM framework (https://blog.yellowant.com/orm-rethinking-data-as-objects-8ddaa43b1410)

- Swagger (https://swagger.io/) i s used to automatically generate an API document (for FastAPI, this plugin is really integrated into the framework).



API Document generated by Swagger

Database

- MySQL (https://www.mysql.com/). Read about the database diagram at: Database Diagram

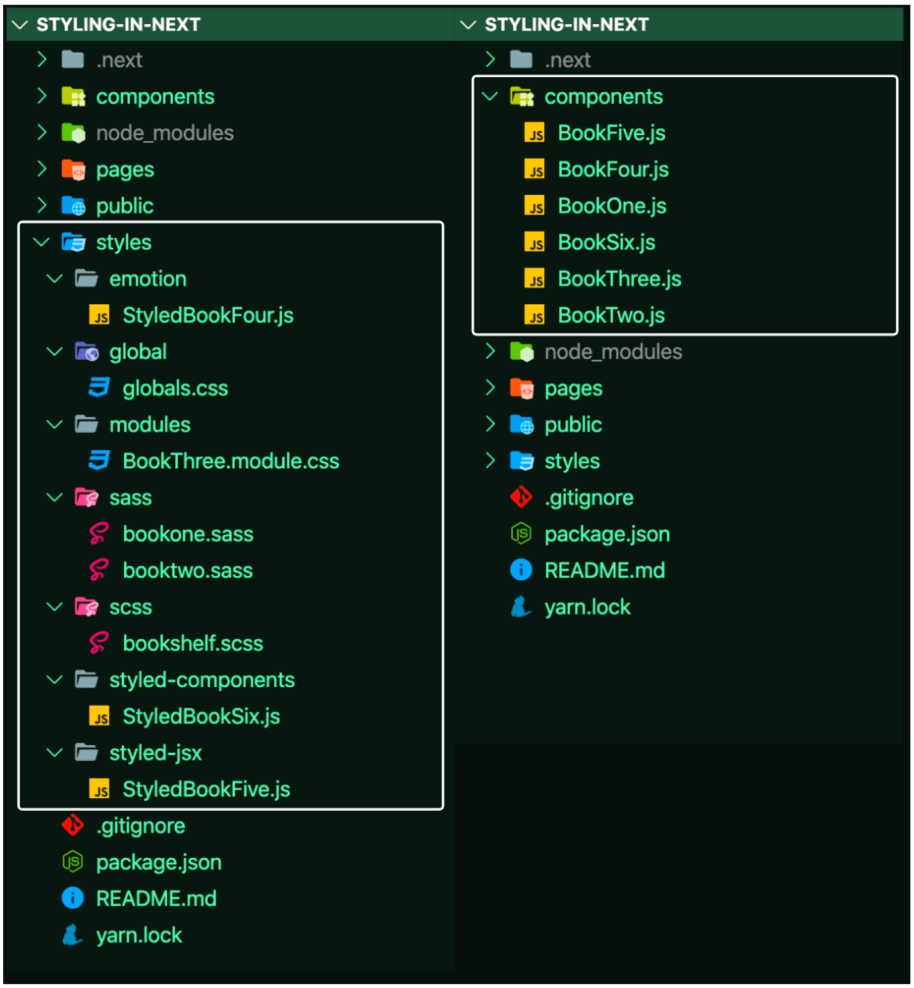
Frontend

- ReactJS (https://reactjs.org/) (version 17 - the latest version). We used the newest programming style of ReactJS that is building functional components with hooks instead of building class components like the last semester's team. This programming style gives us 3 advantages: firstly, writing less code, secondly, making the code more readable, thirdly, easier to debug. The two code snippets below give us a feeling about the difference between using OOP programming and functional programming. The two code blocks do the same thing but the second one is much more readable and shorter.

```
class ClassComponent extends React.Component {
 constructor(props) {
   super(props);
   this.state = {
     count: 0
   };
 }

 render() {
   return (
     <div>
       <p>count: {this.state.count} times</p>
       <button onClick={() => this.setState({ count: this.state.count + 1 })}>
         Click
       </button>
     </div>
   );
 }
}
```

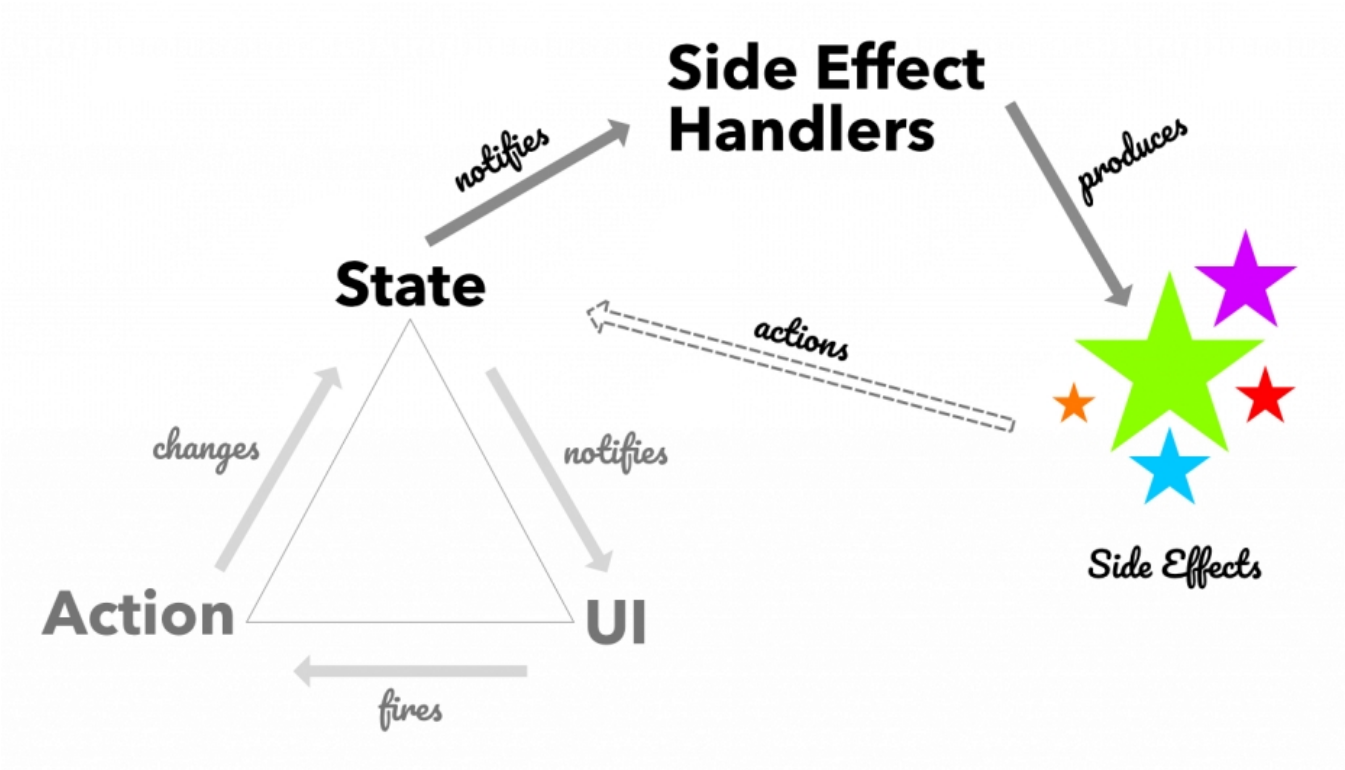
```
const FunctionalComponent = () => {
 const [count, setCount] = React.useState(0);

 return (
   <div>
     <p>count: {count}</p>
     <button onClick={() => setCount(count + 1)}>Click</button>
   </div>
 );
};
```

- For styling the website, instead of using global CSS like the last semester's team, we adopted CSS-IN-JS. With CSS-IN-JS, every component has its own CSS rules, therefore, it makes the JS bundle grow easily. This way of styling is really a perfect combination with Atomic Design (read more about Atomic Design in Frontend Architecture).



Global CSS vs CSS-IN-JS (https://www.smashingmagazine.com/2020/09/comparison-styling-methods-next-js/)

- Ant Design (https://ant.design/. It is a UI library. It makes us more convenient in building UI components for the application.
- For a single page web app, we need a state management library. Instead of using a popular library that is Redux, we decided to use another library that is MobX because we found out that **the performance of MobX is much much better than Redux**. And most importantly, for mobx, we can implement multiple stores for our application instead of only on. This makes the implementation easier. To understand more about MobX, please read at Frontend Architecture.



The side effect model of MobX (Podila et al., 2018)

- Webpack.

Deployment/Hosting

The application is deployed on an AWS EC2 instance, and uses a MySQL database hosted on AWS RDS. The hosting environment has been provided by t he client.

- Docker
- Amazon Web Services (S3, Amplify, EC2, RDS).

Source/Version Control

- GitHub

Testing

- Frontend Testing Framework: Jest + React Testing Library (https://jestjs.io/docs/tutorial-react).
- Backend Testing Framework: pytest (https://pytest.org/).
- Integration Testing: cypress (https://docs.cypress.io/)

Kanban Board

- Trello.

Mockups/Protoypes

- Figma.

Communication

- Slack.