Devlopment of Large Systems & Full Stack Web dev - antik-moderne

Project & Team

Github Project

Alexander Christensen

Emil Aalund

Mikkel René Johansen

Project Management

We adopt the **SCRUM framework** to efficiently manage our development process and quickly respond to changes. We want to use this agile methodology that emphasizes iterative and incremental approaches, allowing flexibility while focusing on delivering customer value.

> This is our attended approach

- Configure a **SCRUM backlog** as a dynamic, prioritized list of features and tasks that evolve as our project progresses. We will be using JIRA to manage this backlog, enabling easy addition, prioritization, and updates.
- **Sprint** fixed period (typically two weeks) during which specific tasks are completed and prepared for review. It is a time-boxed effort.
- **Sprint backlog** consists of tasks selected from the SCRUM backlog for the current sprint, determined during sprint planning. It is managed in **JIRA**, with tasks assigned to the team.
- **Sprint planning** is a meeting at the start of each sprint where the team selects tasks based on priority, goals, and capacity. This sets the direction for the sprint.
- We use **JIRA** to manage both the SCRUM and sprint backlogs. Its features allow us to create user stories, plan sprints, track issues, and report progress, enhancing visibility and collaboration among team members and stakeholders.

Branch Strategy

`Main` ← `develop` ← `feature` ← `developersFeature`

Project – antik-moderne

Our e-commerce platform is a comprehensive online marketplace designed to facilitate the buying and selling of posters across various genres, including animal, art, artists, music, etc. The platform's architecture is informed by key requirements in the areas of functionality, usability, reliability, performance, and supportability. Notably, the platform features user account management, secure payment processing, and robust search functionality, as well as a responsive design and compliance with accessibility standards. The system's reliability is ensured through regular data backups, user-friendly error handling, and a high-availability infrastructure. With a focus on scalability and performance, the platform is designed to support a large user base and high volumes of traffic, making it an ideal case study for exploring the complexities of e-commerce system design and development.

Requirements

Functional Requirements

User Accounts and Product Browsing

- Users should be able to create accounts, log in, and manage their profiles.
- Password recovery and email verification functionalities.
- Users can filter posters by genre, size, price range, and popularity.
- Search functionality with autocomplete suggestions.

Shopping Cart & Checkout Process

- Users can add, remove, and update quantities of posters in their shopping cart.
- Display estimated shipping costs and taxes in the cart.
- Secure payment gateway integration (e.g., PayPal, Stripe).
- Option for guest checkout without account creation.

Order Management

- Users can view order history and track current orders.
- Admin can manage orders, update statuses, and handle returns or exchanges.

Reviews and Ratings & Wishlist Feature

- Users can leave reviews and ratings for posters.
- Admin can moderate reviews to ensure appropriateness.
- Users can save posters to a wish list for future purchases.

Usability Requirements

- The website should be mobile-friendly and accessible on various devices.
- Intuitive navigation and layout for easy browsing.

Reliability Requirements

- Regular backups of user data and product information to prevent data loss.

- User-friendly error messages and logging for system errors.
- Ensure high availability with minimal downtime, ideally 99.9% uptime.

Performance Requirements

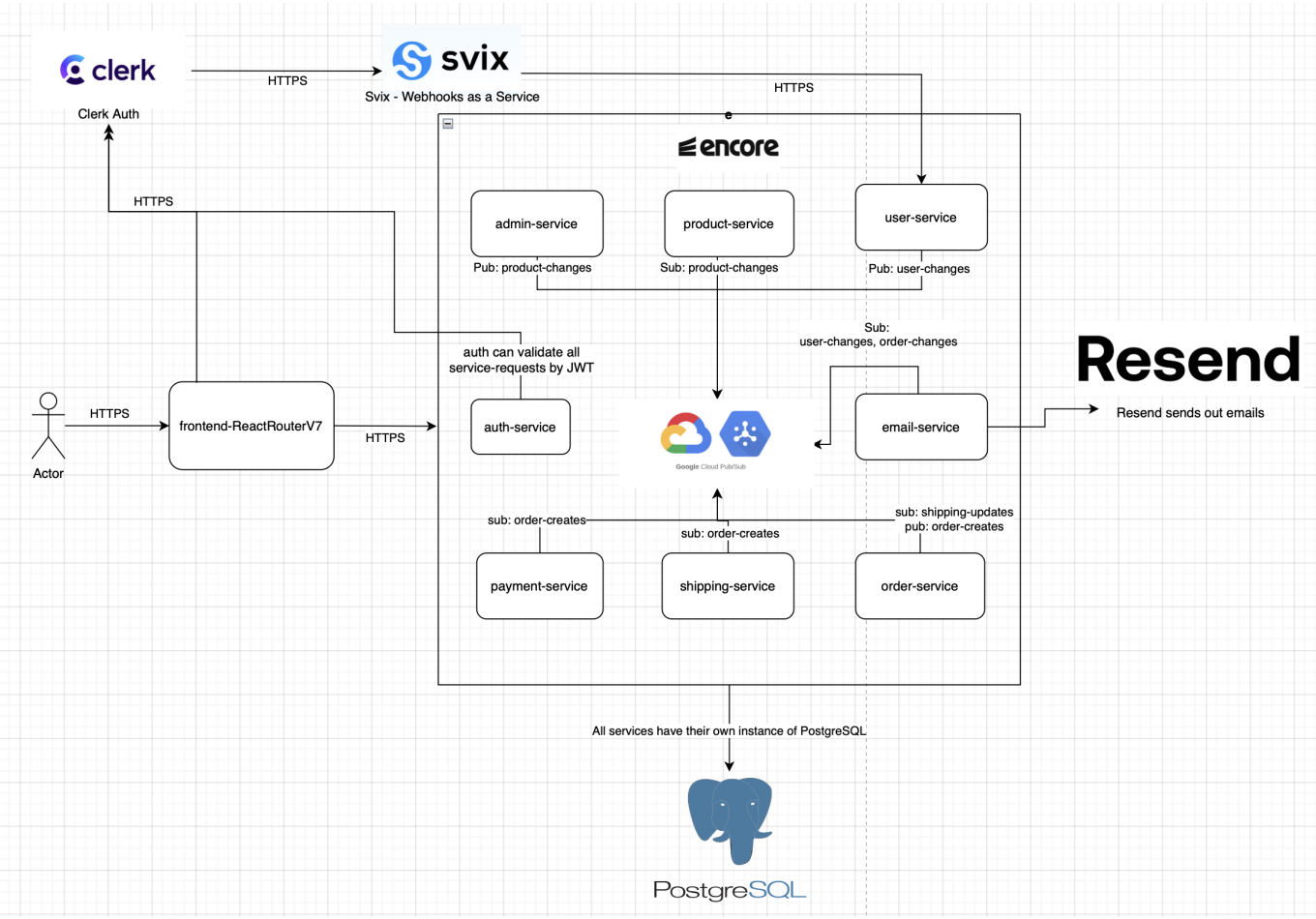
- Pages should load within 2 seconds under normal conditions.
- Support for at least 1000 concurrent users without performance degradation.
- The architecture should allow for easy scaling as the user base grows.

Supportability Requirements

- Comprehensive user manuals and API documentation for developers.
- Integration of a support ticket system and live chat for customer inquiries.
- Implement monitoring solutions to track system performance and user behavior.

System Architecture design

Micro-service architecture



The whole project is organzied as a *mono-repo* with two directories: `backend` & `frontend`. The backend system consists of following micro-services: `admin-service` `auth-service` `email-service` `product-service`.

For the backend **Encore** is used as a backend framework and frontend is builded with React Router v7. Encore is a newer backend development framework designed to streamline creation, deployment and maintenance of cloud-native applications. It's based on a high-performance Rust runtime, *Rust* focus on safety and concurrency enables *Encore* to offer a high-performance runtime environment that is both fast and reliable.

Encore supports *TypeScript* out of the box. This integration enhances code quality and developer productivity by catching errors early in the development process. Further more *Encore* provides a super efficient feature which is this script for frontend *package.json* `"gen": "encore gen client backend-2tui --output=./app/lib/client.ts --env=local"`. This scripts generates all the callable API endpoints for the given services with typesafe input outputs.

Encore provides built-in support for service-to-service communication through its pub/sub system. This feature allows services within the Encore application to publish and subscribe to events, facilitating loose coupling and enhancing the system's scalability and maintainability. This pub/sub is implemented by the use of GCP which is a realtime distributed messaging platform just like *RabbitMQ*, *Kafka* etc.

Key Features of Encore's Pub/Sub System:

- **Type-Safe Communication:** Ensures that messages passed between services adhere to predefined types, improving code reliability and developer experience.
- **Scalable Architecture:** Designed to support a growing number of services and messages, making it suitable for both small and large applications.
- **Decoupled Services:** Services can communicate without direct knowledge of each other, promoting a modular architecture.
- **Built-in Reliability:** Offers mechanisms such as retry policies and dead-letter queues to handle message delivery failures gracefully.

By leveraging Encore's pub/sub system, `antik-moderne` ensures efficient and reliable inter-service communication, laying the foundation for a robust and scalable e-commerce platform.

Authorization outsourced to Clerk

We have choosen to use Clerk for user creation and signup. It can be a true struggle to enhance *SSO / OAuth Protocol / OpenID* on your own and especially *Apple single-sign-on* setup Apple sign in since *Apple SSO* only accepts https connections which mean you have to configure *nginx reverse proxy* for local dev experience. ALI this is Clerk social connection dealing with, and they do also provide custom user handling with the use of webhook calling with Svix - so we are also persisting our users.

Database and use of Typescript ORM Prisma

There exists a lot of different Typescript ORM's but we chosen to work with Prisma since it's proudction ready state and it's support of migrations but up and down.

Commands to configure *Prisma*

```
# Install prisma
npm install prisma --save-dev

# To get the shadow db connection url to Encore.ts shadow database, run:
encore db conn-uri <database name> --shadow

# To initialize Prisma, run the following command from within your service
folder:
npx prisma init --url <shadow db connection url>
```

Encores Prisma docs Encores Prisma github example

Frontend

React Router v7: We've chosen React Router v7 for its dynamic routing capabilities in React applications. This version brings significant improvements in terms of performance and flexibility, allowing us to implement complex routing scenarios with ease. It supports both SSR and CSR, enabling us to optimize our application for performance and SEO.

TypeScript is used throughout our frontend codebase for its strong typing system. This not only helps in catching errors early in the development process but also improves the overall maintainability and readability of the code. TypeScript's compatibility with React and its ecosystem makes it an ideal choice for our project.

TailwindCSS for styling, we've adopted TailwindCSS with its default configuration. TailwindCSS's utility-first approach allows us to rapidly design custom interfaces without stepping away from our code editor. Its configuration-first philosophy means we can customize the framework to fit our design needs perfectly, ensuring consistency and scalability in our design system.

Server-Side Rendering (SSR) & Client-Side Rendering (CSR): Our application supports both SSR and CSR, leveraging the best of both worlds. SSR enhances our application's performance and SEO, serving pre-rendered pages to the browser. CSR, on the other hand, ensures dynamic interactions and a smooth user experience. This dual approach allows us to cater to a wide range of user devices and network conditions, ensuring accessibility and efficiency.

By integrating these technologies, our frontend architecture is designed to be flexible, efficient, and scalable, supporting the complex needs of our e-commerce platform while providing an excellent developer experience.

Dev environment

Since we use **Encore** as backend framework, encore have a super efficient and elegant dev-experience to boot up whole backend system (dev-exp for frontend is described below):

```
# Given you're placed in root
cd backend

# If first time whole project have to have needed dependencies installed
npm install

# given you have configured encore -> https://encore.dev/docs/ts/install
encore run
```

Frontend application for dev-exp:

```
# Given you're placed in root

cd frontend

# If first time whole project have to have needed dependencies installed
npm install

npm run dev
```

If you gonna experiement with Clerk usage of Webhooks:

```
# Ensure you have instaled ngrok
ngrok http --url=sharp-moth-exciting.ngrok-free.app 4000
```

Bonus links and miscellaneous

PostgreSQL is used as databasse - where *Drizzle* is used as *ORM*.

Encore example with PostgreSQL and Drizzle as ORM

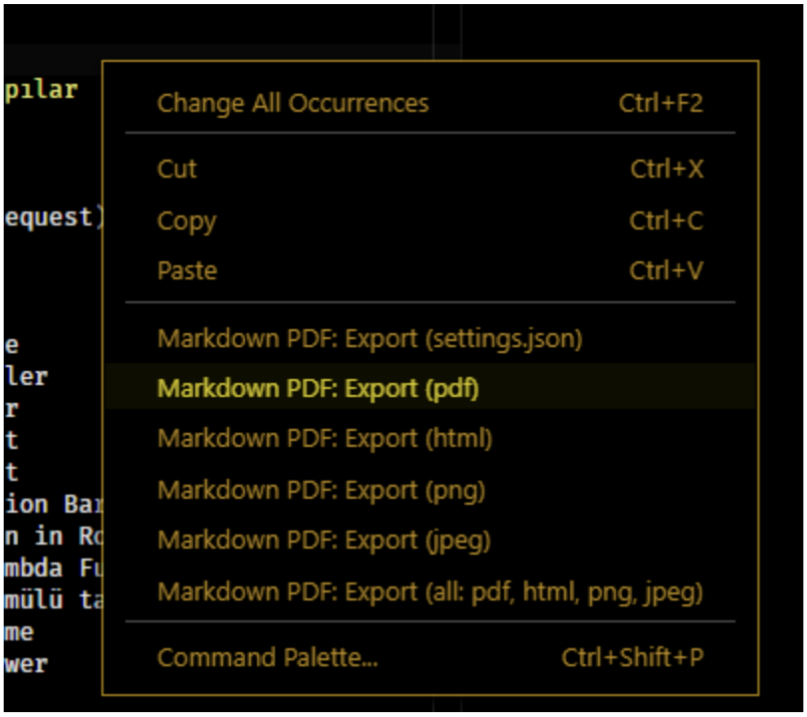
Clerk React SDK + Encore App Example

Stackoverflow about issues with migrating to Tailwindcss v4

React spinners components

Via a Visual Studio Code extension (tested in 2020)

- Download the Yzane Markdown PDF extension
- Right click inside a Markdown file (`md`)
- The content below will appear
- Select the `Markdown PDF: Export (pdf)` option



NOTES

For the authentication following is used: **clerk**, **svix** & **ngrok**. Clerk is handling the signin with either google / apple or manual signup. *Svix* is what *Clerk* uses for their webhooks.

Guide for configuring webhooks

Using *ngrok* for exposing local running encore services where /users/webhook is relevant to make webhook listening for incomming requests when user: **creates**, **updates** or **deletes**.

ngrok to expose local running endpoints used for clerk/svix webhook when new users or users updates or deletes execute this to expose: `ngrok http --url=sharp-moth-exciting.ngrok-free.app 4000`