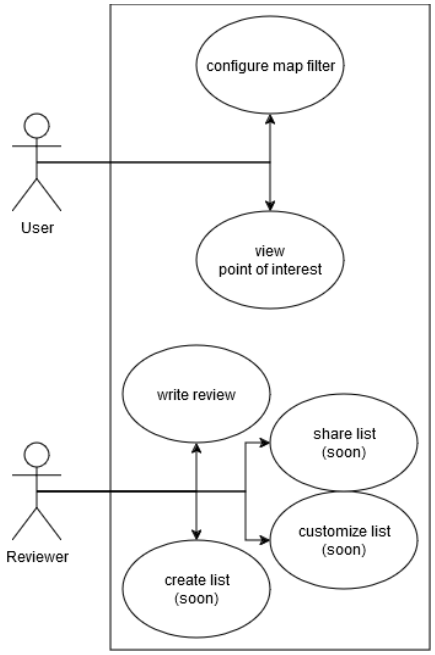
Overall Use Case Diagram



Overall Use Case Diagram – Hebt hervor was die Schwerpunkte eurer Entwicklung (Demo) sind

1. Software Tools

- **IDEs**: Visual Studio Code, IntelliJ IDEA
- **Versionierung**: GitHub
- **Testing-Tools**: Postman
- **Projektmanagement**: Jira
- **Docs**: Python, MkDocs

2. Plattformen

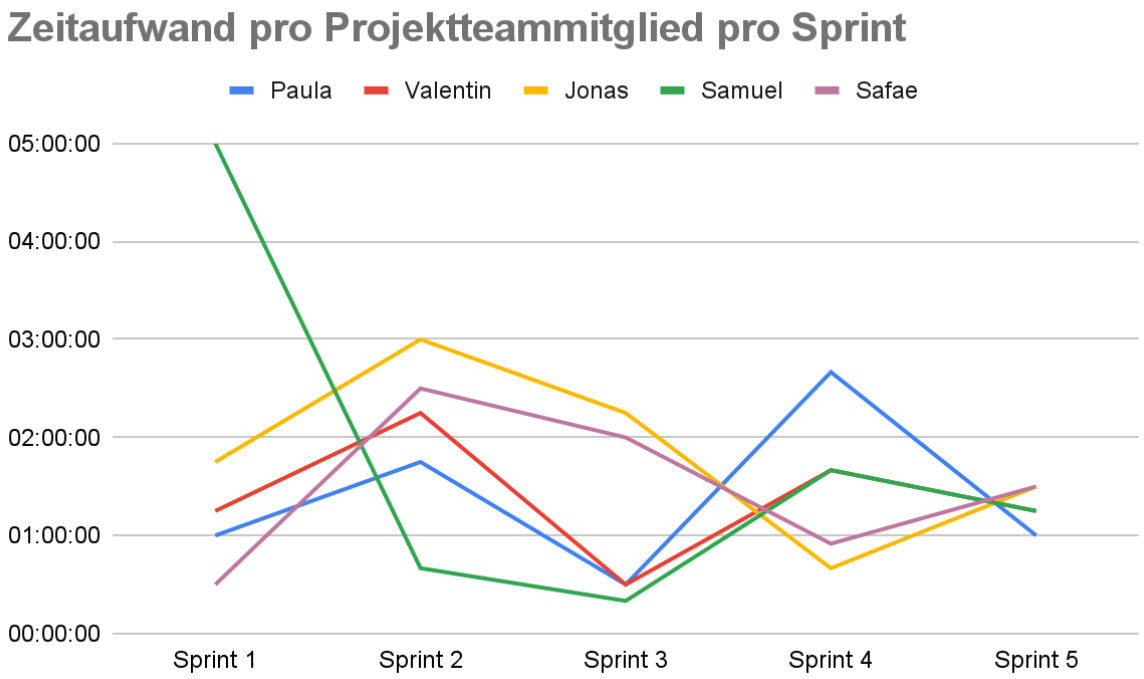
- **Hosting**: Docker
- **Frameworks**: Vue.js (JavaScript), Spring Boot (Java), Docker Compose

3. Techniken:

- **Programmierparadigmen**: Objektorientiert, funktional, reaktiv.
- **Entwicklungsmethoden**: Agile, Scrum, Kanban
- **Spezielle Patterns**: MVC, MVVM, Repository Pattern.

4. Zeitaufwand:

Stunden pro Person

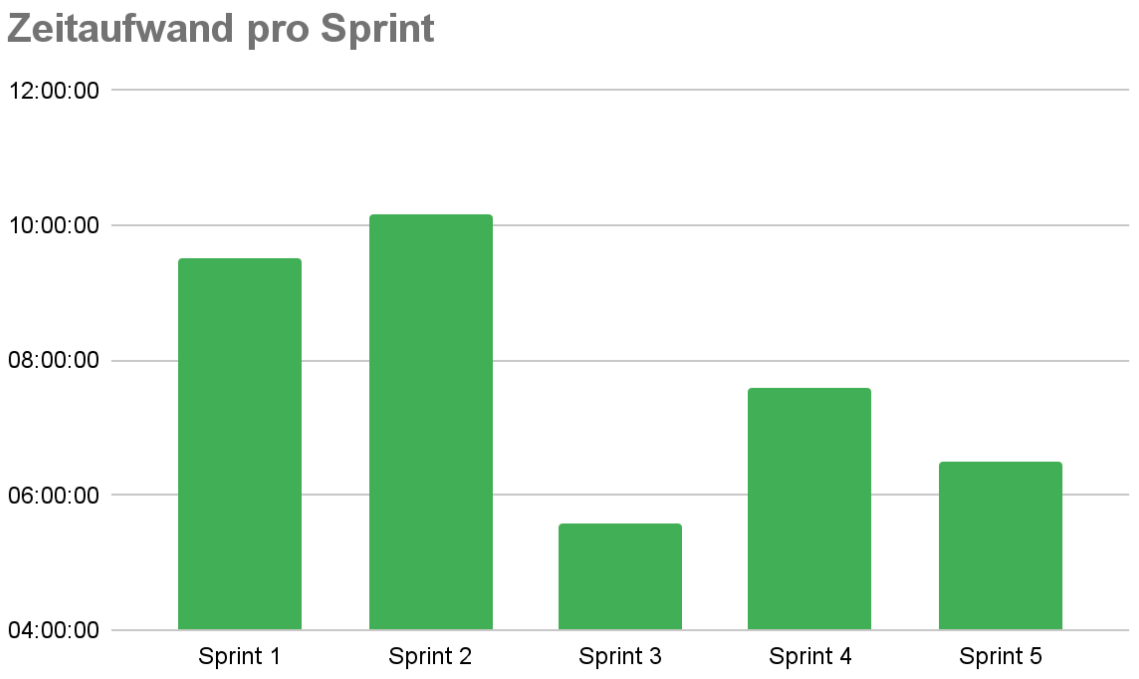


Hauptbeitrag pro Person

Person	Hauptbeitrag
Safae K.	Spring Boot Backend Initialisierung
Jonas S.	Frontend
Valentin W.	Openstreetmap Backend Anbindung
Paula K.	Design System in Figma erstellen
Samuel B.	Blogeinträge & Dokumentation

Stunden pro Workflow

Workflow	Requirement Analysis	Projektmanagement	Design	Entwicklung
Zeitaufwand (in Stunden)	2,5	5,3	3,5	4



5. Architekturstile/-entscheidungen:

1. Layered Architecture (Schichtenarchitektur)

Frontend und Backend:

- Klare Trennung der Schichten:
- Frontend (Vue.js) für die Benutzeroberfläche und Interaktionen.
- Backend (Spring Boot) für Geschäftslogik und Datenverarbeitung.
- Datenbank (PostgreSQL) zur Datenspeicherung.

Vorteile:

- Separation of Concerns erleichtert die Wartung.
- Jede Schicht erfüllt eine spezifische Aufgabe (z. B. Benutzerverwaltung im Backend)

2. Event-Driven Architecture

Verwendung von GitHub Actions (später) für automatisierte CI/CD-Prozesse.
JWT-Authentifizierung (später) (JSON Web Token) im Backend:

- Authentifizierungsereignisse triggern Sicherheitsprüfungen und Token-Generierungen.

Interaktionen zwischen Frontend und Backend:

- Benutzeraktionen (z. B. Filtern von Restaurants) lösen API-Aufrufe und Backend-Operationen aus

3. Microservices-Prinzipien (Ansätze)

Modularität:

- Komponenten wie Benutzerverwaltung, Review-Management und OSM-Integration sind klar abgegrenzt und könnten unabhängig weiterentwickelt werden.
- Ermöglicht zukünftige Erweiterungen ohne größere Umstrukturierung.
- Datenbank und APIs agieren als unabhängige Ressourcen, die von verschiedenen Teilen des Systems genutzt werden

4. API-Driven Architecture

Die OSM-Integration (OpenStreetMap API) zeigt einen API-zentrierten Ansatz:

- Frontend sendet spezifische Filterparameter über das Backend an die API.
- Die Architektur entkoppelt die Interaktion mit der OSM-API, um Sicherheitsrisiken zu minimieren und die Wartbarkeit zu erhöhen.

5. Component-Based Architecture

Vue.js-Frontend:

- Modulbasierter Aufbau ermöglicht die einfache Entwicklung und Erweiterung einzelner UI-Komponenten (z. B. Filteransicht, Kartenansicht, Restaurantprofile).

Spring Boot-Backend:

- Verwendung des MVC-Musters mit getrennten Controllern, Services und Modellen erhöht die Übersichtlichkeit und Skalierbarkeit

.

Schlüsselaspekte und Argumente

Modularität und Erweiterbarkeit:

- Die Architektur ist flexibel und erleichtert zukünftige Funktionserweiterungen wie die Listenfunktion und erweiterte Filteroptionen.

Sicherheit:

- JWT-Token und ORM schützen die Datenintegrität und verhindern Sicherheitslücken wie SQL-Injections.

Skalierbarkeit:

- Dank API-Integration und modularer Trennung können Lastspitzen effizient gehandhabt werden.