# DV-Konzept zur Anwendung Sündenbock- Gruppe 11

## Vorgehen

Das Projektvorgehen orientiert sich am agilen Projektmanagement, insbesondere an der Methode des Minimal Viable Product (MVP). Das MVP definiert einen Kern von "Must-Have"-Anforderungen, mit dem Ziel, schnellstmöglich eine produktive und lauffähige Software bereitzustellen.

Weitere Funktionalitäten und "Should-Have"-Anforderungen werden anschließend in iterativen Releases hinzugefügt. Das leitende Prinzip der Entwicklung ist das Mantra "Try-Fast-Fail-Fast", um frühzeitig Feedback zu sammeln, Annahmen zu validieren und den Entwicklungsprozess entsprechend anzupassen.

### Realisierung des MVP

Um dieses Ziel zu erreichen, wurde das MVP als Durchstich realisiert, der alle Anwendungsschichten umfasst. Das Vorgehen konzentrierte sich darauf, zunächst das technische Fundament zu legen und anschließend die Kernfunktionalität End-to-End zu implementieren:

- **Basis-Setup und Backend-Fokus:**
  - Zuerst wurde die Projektstruktur als Monorepo (Backend/Frontend) aufgesetzt und die grundlegende Kommunikation (z. B. CORS-Settings) konfiguriert.
  - Anschließend lag der Fokus auf dem Backend, um einen ersten "Durchstich" mit Dummy-Daten zu implementieren und die API-Dokumentation (mittels Swagger/OpenAPI) bereitzustellen.
- **Frontend-Anbindung und Kernfunktionalität:**
  - Aufsetzend auf dem funktionalen Backend-Gerüst wurde der Frontend-Durchstich gestartet. Dies umfasste den Aufbau der Frontend-Architektur und die erste Visualisierung von Daten der Getter-Endpunkte (mittels DaisyUi).
  - Parallel wurde die zentrale "Must-Have"-Anforderung – die Authentifizierung – implementiert. Dies beinhaltete das State Management für den User-Login (JWT) und die Definition eines Rollen- und Rechtekonzepts (Admin, Developer, User).

- In diesem Zuge wurden auch die Verantwortlichkeiten für die Ausführung von Geschäftslogik (z.B. Validierungen im Frontend vs. Backend) festgelegt.

## Iterative Weiterentwicklung

Auf Basis dieses lauffähigen MVP (bestehend aus Fundament, API-Durchstich und Authentifizierung) werden alle weiteren Funktionalitäten und "Should-Have"-Anforderungen (z.B. Ticket-Verwaltung, Kommentarfunktionen) in anschließenden iterativen Zyklen entwickelt und hinzugefügt.

## Architektur und Technologie-Stack

Die Anwendungsarchitektur basiert auf einem **Monorepo-Ansatz**, der Backend und Frontend in einem gemeinsamen Repository verwaltet, um Abhängigkeiten und die gemeinsame Entwicklung zu vereinfachen.

Die Architektur des Backends basiert auf dem **Spring Boot Framework**. Diese Entscheidung ermöglichte eine schnelle Konfiguration und eine Dependency-Management-Struktur, die auf die Erstellung produktionsreifer Anwendungen ausgelegt ist. Die Auswahl der zentralen Technologien, die in der Maven-Konfigurationsdatei pom.xml deklariert sind, erfolgte mit dem Ziel, eine moderne, sichere und wartbare **3-Schichten-Architektur** gemäß den Anforderungen der Vorlesungsunterlagen zu realisieren. Unsere technologische Basis umfasst:

- **Spring Boot:** Dient als Basis-Framework für Dependency Injection, Konfiguration und als Webserver.
- **Spring Data JPA & Hibernate:** Nutzen wir für unsere Persistenzschicht. Spring Data JPA vereinfacht die Erstellung von Repositories, während Hibernate als ORM unsere Java-Objekte auf Datenbanktabellen abbildet.
- **Spring Security:** Bildet das Fundament für unsere gesamte Authentifizierung und Autorisierung.
- **JSON Web Token (JWT):** Wir verwenden die jjwt-Bibliotheken zur Erstellung und Validierung von JWTs für unsere zustandslose Authentifizierung.
- **H2 Database:** Dient uns als leichtgewichtige In-Memory-Datenbank für die Entwicklung und unsere Tests.
- **MapStruct:** Setzen wir für das performante Mapping zwischen unseren Daten-Entitäten und DTOs ein.
- **Springdoc OpenAPI (Swagger):** Stellt automatisch eine interaktive API-Dokumentation basierend auf unseren Controllern und Annotationen bereit.
- **JUnit 5, Mockito & MockMvc:** Bilden unser Test-Framework für Unit- und Integrationstests.

Das Frontend basiert auf **Angular (Version 20)**. Es weicht von einer klassischen, schichtenbasierten Struktur (z. B. globale Ordner für services, pages, components) ab. Stattdessen orientiert sich die Architektur stark an **fachlichen Domänen** und folgt einem **Feature-basierten Ansatz**.

Für die Gestaltung der Benutzeroberfläche wird **Tailwind CSS** als Utility-First-Framework verwendet, ergänzt durch die Komponentenbibliothek **DaisyUI**, die eine schnelle Entwicklung von konsistenten UI-Elementen auf Basis von Tailwind ermöglicht.

Passend zum agilen Vorgehen wird die Anwendung in "Features" (Funktionsmodule) gegliedert. Jedes Feature-Modul (z. B. auth, project, ticket) bündelt alle zugehörigen Komponenten, Services und State-Management-Logiken. Diese domänenspezifische Ordnerstruktur soll die Modularität, Kapselung und Wartbarkeit fördern.

**Weitere zentrale Technologien und Konzepte:**

- **Reaktives State Management:** Im Frontend wird für ein modernes und reaktives State Management eine Kombination aus **RxJS** und **Signals** verwendet.
- **Containerisierung:** Die Anwendungskomponenten (Backend, Frontend) werden als **Docker-Images** bereitgestellt, um eine konsistente Ausführungsumgebung in Entwicklung, Test und Produktion sicherzustellen.
- **CI/CD:** Eine **CI/CD-Pipeline** (Continuous Integration / Continuous Deployment) wird eingesetzt, um den Prozess von Code-Änderungen bis hin zum Deployment zu automatisieren, inklusive Build, Test und Auslieferung der Docker-Images

## Zentrale Geschäftsobjekte

Die fachliche Domäne des Fehlerverfolgungssystems wurde modelliert, um die zentralen Entitäten und ihre Beziehungen zueinander abzubilden. Das vollständige **UML-Klassenmodell** der zentralen Geschäftsobjekte (u. a. User, Ticket, Project, Comment) ist unter *UML-Klassenmodell der zentralen Geschäftsobjekte.pdf* zu finden.

## Dialogmodell

Das Dialogmodell skizziert die wesentlichen Maskenlayouts und visualisiert die Navigationsflüsse zwischen den einzelnen Ansichten der Anwendung. Es stellt den Weg des Benutzers von der Authentifizierung (Login/Registrierung) über die Projekt- und Ticketübersicht bis hin zur Detailansicht eines Tickets dar. Es ist unter Dialogmodell_Gruppe11.png zu finden.

## Wesentliche Testfälle

Zur Sicherstellung der Kernfunktionalitäten wurden wesentliche Testfälle definiert. Diese decken die Hauptanwendungsfälle (Use Cases) wie Authentifizierung, Ticketerstellung und -verwaltung sowie die Kommentarfunktion ab und sind unter *Testcases_Suendenbock_Gruppe11.csv* zu finden.

Während der Konzeptionsphase wurden spezifische technische und fachliche Annahmen mit den Dozenten (Auftraggeber) abgestimmt, um die Einhaltung der Anforderungen und Bewertungskriterien sicherzustellen.

# Annahmen/Abstimmungen mit Auftraggeber (Dozenten)

## Rollen- und Rechtekonzept (RBAC)

Da das Lastenheft kein detailliertes Rechtekonzept spezifizierte, wurde ein Role-Based Access Control (RBAC) Ansatz vorgeschlagen.

- Entscheidung: Der Ansatz wurde als sinnvoll bewertet.
- Umsetzung (MVP): Es wird eine feste Liste von Rollen (Admin, Developer etc.) mit vordefinierten Rechten implementiert.

## Bestätigte Sicherheitsstandards

Zwei zentrale Sicherheitskomponenten wurden validiert:

- Passwort-Hashing: Die Verwendung des BCryptPasswordEncoder wurde als adäquater Standard bestätigt.
- Authentifizierung (JWT): Der gewählte Ansatz einer zustandslosen Authentifizierung mittels JSON Web Tokens (JWT) wurde als professionelle, den Kriterien entsprechende Lösung bestätigt.

## Technologie-Stack und Konventionen

- Moderne Versionen: Die Verwendung aktueller Technologie-Versionen (z.B. OpenJDK 25, Angular 20) wurde explizit genehmigt und als positiv bewertet.
- Frontend-Bibliotheken: Die Grundanforderung ist Angular. Der zusätzliche Einsatz nützlicher Bibliotheken (wie Daisy UI, Tailwind, RxJS, Signals) wurde als sinnvoll bestätigt und wird bei korrekter Nutzung positiv bewertet.

- Framework-Konventionen: Es wurde bestätigt, dass die Einhaltung aktueller Framework-Standards korrekt ist, auch wenn diese von älteren Beispielen abweichen.

## Genutzte Hilfsmittel und Verwendungszweck

| Hilfsmittel | Verwendungszweck |
|---|---|
| **Gemini Pro** | Formatierung, JavaDoc, Hilfe bei Strukturaufbau, Erklärung und Verständnis von Konzepten, Code Optimierung, Testfälle |
| **Chat GPT** | Formatierung, Docs, Hilfe bei Strukturaufbau, Erklärung und Verständnis von Konzepten, Code Optimierung |
| **GitHub Copilot** | JavaDoc, Fixes, Explain code |
| **https://www.w3schools.com/java/default.asp** | Java Basics |
| **https://www.baeldung.com/java-json-web-tokens-jjwt** | JWT-Verständnis |
| **https://medium.com/@victoronu/implementing-jwt-authentication-in-a-simple-spring-boot-application-with-java-b3135dbdb17b** | JWT-Verständnis |
| **https://www.baeldung.com/security-spring** | Spring security verständnis |
| **https://mapstruct.org/documentation/stable/reference/html/** | MapStruct verständnis |
| **https://docs.spring.io/spring-framework/reference/core/expressions.html** | SpEL und @PreAuthorize verständnis |
| **https://medium.com/@vishamberlal/understanding-data-transfer-objects-dto-in-spring-boot-ac06b575a1d5** | DTO Verständnis |
| **https://www.baeldung.com/exception-handling-for-rest-with-spring** | Exception Handling |
| **https://daisyui.com/** | Styling basic Komponenten UI |
| **https://www.typescriptlang.org/docs/** | Typescript Basics und mehr |
| **https://angular.dev/overview** | Verständnis Signals, ReactiveForms, Routing, HTTP Client, Components |