Ergebnisbericht Übungsblatt 2 SA4E

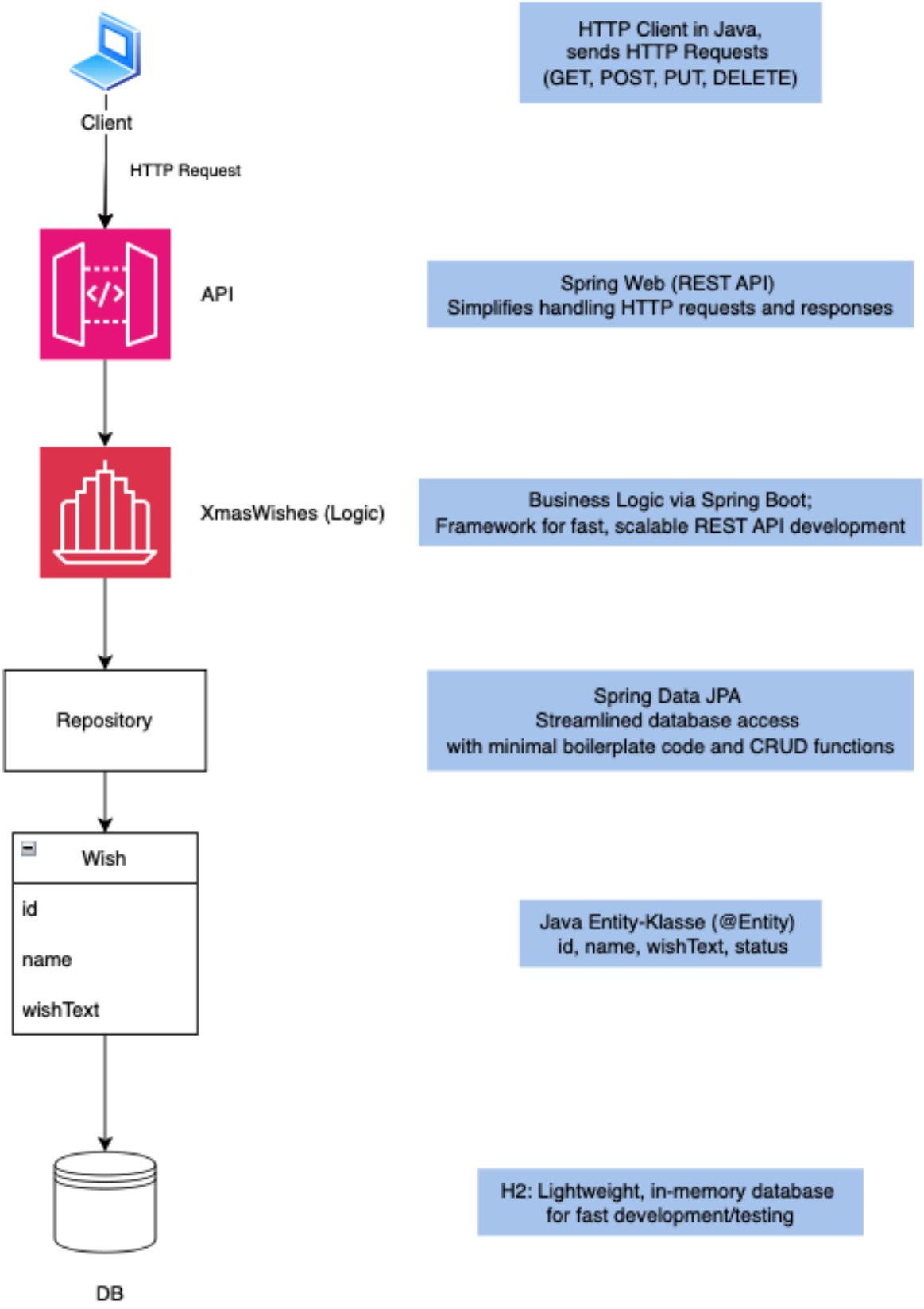
Simon Haebenbrock, s4sihaeb@uni-trier.de, dc: SimonH (simon_7672)

Für die Entwicklung der Anwendung wurde Java 17 verwendet, die Entwicklung fand in IntelliJ IDEA statt, und Apache Maven kam für das Build-Management und die Verwaltung der Abhängigkeiten zum Einsatz.

1. Architektur-Design
In der ersten Phase habe ich das System XmasWishes modelliert, um sicherzustellen, dass es die Anforderungen an Skalierbarkeit und Verfügbarkeit erfüllt. Die Architektur musste so gestaltet sein, dass sie auch mit einer potenziell hohen Anzahl an Anfragen pro Sekunde umgehen kann, die im letzten Quartal des Jahres auftreten. Die Lösung setzt auf Microservices, bei denen jeder Microservice für spezifische Aufgaben zuständig ist.

Ich habe folgende Komponenten vorgesehen:

- **Client:** Einfache Clients (z. B. Web- oder mobile Anwendungen), die HTTP-Anfragen an das System senden, um Wünsche zu speichern.
- **API:** Eine RESTful API, die die Interaktion zwischen den Clients und der Logik des Systems ermöglicht. Diese API sorgt für die Skalierbarkeit des Systems.
- **XmasWishes-Logik:** Dies stellt den Kern des Systems dar, der die Geschäftslogik umsetzt. Hier wird der Status eines Wunsches gespeichert und verwaltet.
- **Datenbank:** Eine relationale Datenbank, die alle Daten speichert. Um die Skalierbarkeit zu gewährleisten, wurde eine einfache Lösung wie H2 für das Testing verwendet.
- **Repository:** Diese Komponente stellt eine Schnittstelle für den Zugriff auf die Datenbank bereit und nutzt Spring Data JPA für effizienten Datenzugriff.


Client
HTTP Request
API
XmasWishes (Logic)
Repository
Wish
id
name
wishText
DB
HTTP Client in Java,
sends HTTP Requests
(GET, POST, PUT, DELETE)
Spring Web (REST API)
Simplifies handling HTTP requests and responses
Business Logic via Spring Boot;
Framework for fast, scalable REST API development
Spring Data JPA
Streamlined database access
with minimal boilerplate code and CRUD functions
Java Entity-Klasse (@Entity)
id, name, wishText, status
H2: Lightweight, in-memory database
for fast development/testing

2. Detaillierung der Softwaretechnologien
Ich habe folgende Softwaretechnologien ausgewählt:

- **Spring Boot**: Für die Implementierung der REST API und der Geschäftslogik, da es eine schnelle Entwicklung von Microservices ermöglicht.
- **Spring Data JPA**: Für den Zugriff auf die Datenbank, um die CRUD-Operationen zu vereinfachen und die Wartbarkeit des Codes zu erhöhen.
- **H2-Datenbank**: Eine leichtgewichtige, in-memory Datenbank, die für Tests und Entwicklungszwecke verwendet wird. In einer produktiven Umgebung könnte eine skalierbare Datenbank wie PostgreSQL eingesetzt werden.
- **Spring Web**: Für die Handhabung von HTTP-Anfragen und -Antworten.

An dieser Stelle sei schon einmal erwähnt, dass die Spring Boot-Anwendung mit Controller und Repository noch keine vollständige, produktionsreife API darstellt. Sie bildet vielmehr ein einfaches Backend, das für die Demonstration der grundlegenden Geschäftslogik und Dateninteraktion entwickelt wurde. Eine voll funktionsfähige API würde zusätzlich Sicherheitsmechanismen, eine ordnungsgemäße Fehlerbehandlung, Authentifizierung, Autorisierung und möglicherweise eine API-Dokumentation erfordern. Das aktuelle Setup dient primär der Veranschaulichung des Funktionsumfangs, ohne die komplexeren Anforderungen einer ausgereiften API-Umgebung zu erfüllen.

Die API basiert auf REST und kommuniziert mit einem Frontend, das HTTP-Anfragen an das System stellt. Das Backend verwendet Spring Boot, um die Geschäftslogik zu implementieren, einschließlich der Verwaltung des Wünschesstatus und der Speicherung der Daten.

3. Prototyp
Für den Prototypen habe ich eine einfache Anwendung entwickelt, die die grundlegenden Funktionen des Systems implementiert, einschließlich der Verwaltung von Wunschdaten. Die Anwendung ermöglicht es, Wünsche zu speichern, deren Status zu ändern und zu prüfen.
Die Implementierung ist so gestaltet, dass sie leicht skalierbar ist. Ich habe die API mit Spring Boot und Spring Data JPA entwickelt, um die Verbindung zur Datenbank zu erleichtern. Die Kommunikation mit der Datenbank erfolgt über eine Entity-Klasse, die die Datenstruktur für den Wunsch speichert.
Die API implementiert die grundlegenden HTTP-Methoden:

- **GET:** Abrufen eines Wunsches.
- **POST:** Hinzufügen eines neuen Wunsches.
- **PUT:** Aktualisieren eines bestehenden Wunsches.
- **DELETE:** Löschen eines Wunsches.

Das Ergebnis des API-Performance-Tests zeigt, dass das Backend bei 10.000 parallelen Anfragen eine beachtliche Performance von rund 3.160 Anfragen pro Sekunde erzielt. Dabei ist jedoch zu beachten, dass dieser Test auf einer einfacheren Backend-Struktur basiert, die keine umfassenden Produktionsmerkmale wie Fehlerbehandlung, Authentifizierung und Validierung enthält. Die 0% Fehlerrate in diesem Test ist also nicht repräsentativ für die reale Ausfallsicherheit in einer komplexeren Umgebung.

In Bezug auf die in Teilaufgabe 1 festgelegten Leistungsregionen ist es nicht zu erwarten, dass der getestete Ansatz diese Zielwerte erreichen kann, da er nicht auf einer voll ausgereiften und optimierten Infrastruktur basiert. Allerdings verdeutlicht der Test, dass der Backend-Ansatz durchaus skaliert, sodass durch einen Mehreinsatz der Hardware — etwa durch zusätzliche Server oder optimierte Lastverteilung — das Nachrichtenvolumen weiter gesteigert werden könnte. Ein solcher Ansatz würde es ermöglichen, mit wachsendem Datenaufkommen umzugehen und die Performance auch in einer produktionsnahen Umgebung weiter zu verbessern.

4. Apache Camel Lösung

Für Teilaufgabe 4, die Implementierung der Apache Camel Lösung, habe ich eine Camel-Route entwickelt, die die gescannten Wunschdaten aus einer Datei einliest, an das XmasWishes-System weitergibt und den Wunschstatus aktualisiert. Leider konnte die vollständige Integration in die bestehende Spring Boot-Anwendung aufgrund technischer Schwierigkeiten nicht realisiert werden. Das Problem lag bei der Integration von Camel in die Spring Boot-Anwendung, wodurch das Lauschen und Verarbeiten der Datenströme nicht wie erwartet funktionierte.

Die Route und der File Generator wurden erfolgreich implementiert, jedoch scheiterte die eigentliche Verarbeitung der Daten innerhalb der Spring Boot-Umgebung. Die Integration selbst konnte daher nicht abgeschlossen werden.

Quellen:

- Spring Boot Documentation. https://spring.io/projects/spring-boot
- Apache Camel Documentation. https://camel.apache.org/manual/latest/
- GitHub Copilot zur Generierung von Boilerplate-Code