

Web Engineering 3

Vorlesung 1

Hochschule Zittau/Görlitz

Christopher-Manuel Hilgner

Organisatorisches

- Vorlesungen + Seminar
- Projektarbeit im Laufe des Semesters
- Gruppenarbeit möglich für das Projekt
- Crossbeleg mit anderen Modulen wird empfohlen
- Beleg mit Verteidigung am Ende des Semesters
- Zur Abgabe gehören:
 - Beleg als PDF
 - Folien der Verteidigung
 - Alle Projektdaten, in einem Git Repository

Beleg

- Keine festgelegte Seitenanzahl (bei mir waren es damals 10-15 Seiten)
- Anforderungsanalyse
- Beschreibung der Umsetzung
- Gründe für Entscheidungen bei der Entwicklung darstellen
- Dokumentation der Software

Verteidigung

Dauer: ca. 20 Minuten

- Demonstration des finalen Projekts
- Umsetzung Vorstellung
 - Technologien
 - Struktur
- Kurz ausgewählte Programmbestandteile vorstellen, die als wichtig angesehen werden

Material

<https://github.com/XPromus/we3-hszg>

- Script, Folien, Beispielprojekte

Inhalte

- Entwicklung von Full-Stack Web Anwendungen
- Spring für Backend Entwicklung
- Docker/Podman zum Deployen von Anwendungen
- Frontend Frameworks wie Svelte, Vue oder React...
- Debugging Tools und Techniken
- Testing
- Authentication

Grundbegriffe

Client Side

- Alles mit dem der Nutzer interagiert
- Gute UI und UX wird benötigt für einfach Interaktion

Server Side

- Alles was auf dem Server passiert
- Funktionen werden dem Frontend bereitgestellt

API

- Das Frontend und Backend kommunizieren über APIs
- Erlauben Anwendungen in Schichten zu unterteilen
- Können der Middleware zugeordnet werden

Grundbegriffe

REST

- Architektur Stil für Kommunikation in Web Systemen
- Separation von Server und Client
- Komplett Stateless: Server und Client brauchen keine Kontext Informationen über den jeweils anderen
- Kommunikation findet über Requests und Responses statt
- Alle CRUD Operationen werden direkt auf HTTP Methoden gemapped

Grundbegriffe

MVC - Model-View-Controller

- Design Pattern zur Implementation von User Interfaces, Daten und Kontroll Logik
- Fokus auf Separierung von Business Logik und visueller Darstellung

Komponenten:

- **Model:** Verwaltet Daten und Business Logik
- **View:** Übernimmt Layout und Darstellung
- **Controller:** Leitet Befehle zum Model und View weiter

Grundbegriffe

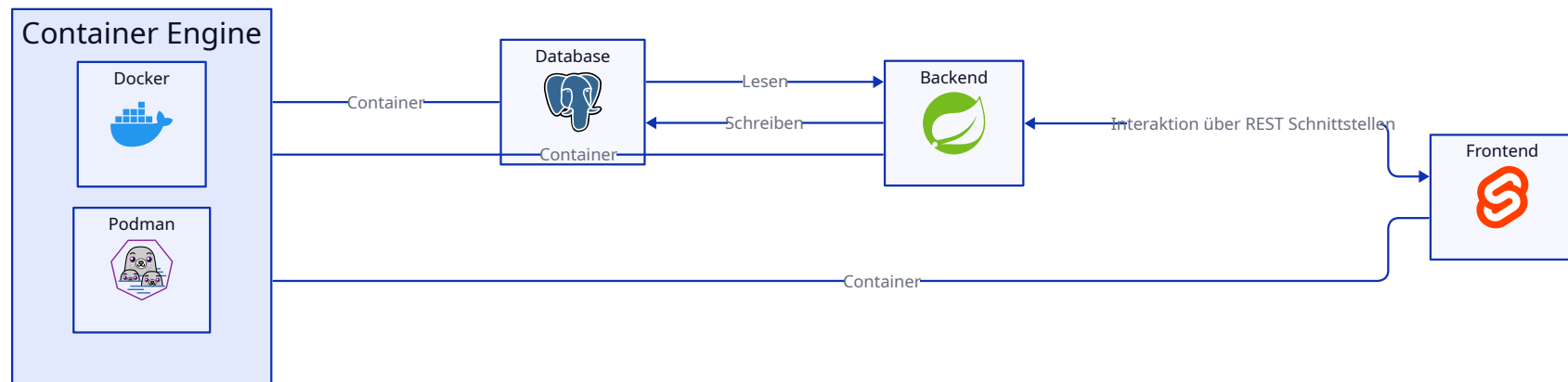
CRUD

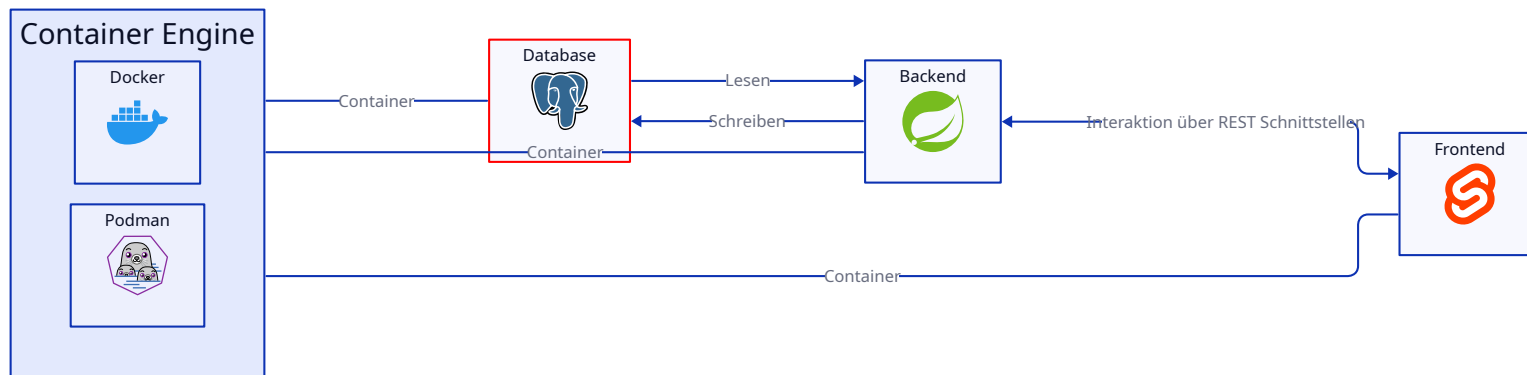
- Vier Grundoperationen um mit persistenten Daten zu interagieren
- **Create**: Neue Daten Einträge werden gespeichert.
- **Read**: Existierende Daten werden ausgelesen
- **Update**: Existierende Daten werden aktualisiert
- **Delete**: Daten werden gelöscht

ORM - Object-Relational-Mapping

- Technik um relationale Datenbank in Objekte einer Programmiersprache umzuwandeln
- Erstellung einer virtuellen Objekt Datenbank
- ORM Frameworks erlauben durchführen von CRUD Operationen
- SQL-Befehle müssen nicht per Hand geschrieben werden

Struktur einer Full-Stack Web App

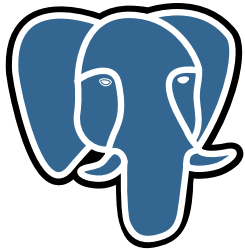


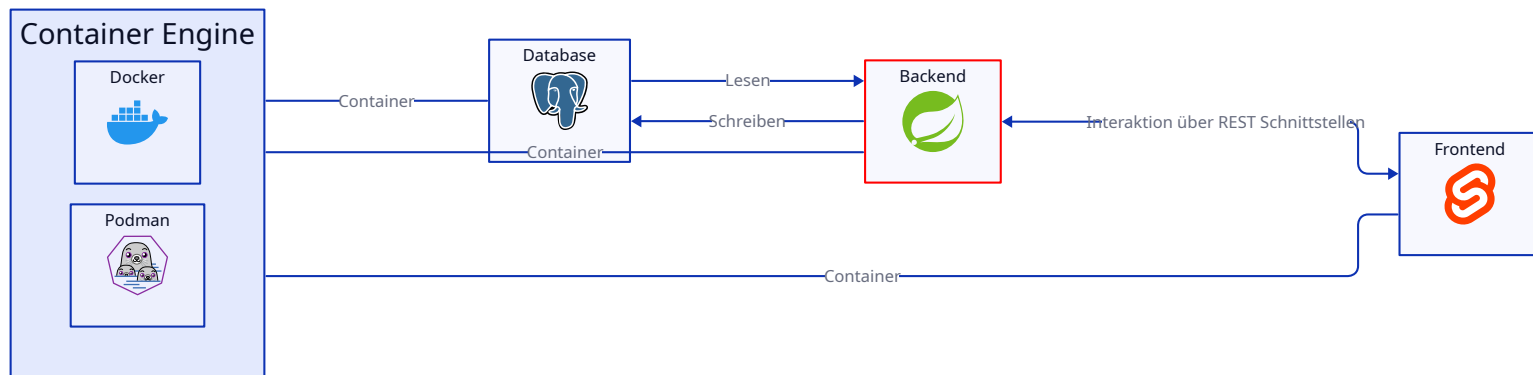


Database

- Enthält alle zu speichernden Daten
- Manuelle Erstellung bzw. Konfigurierung nicht nötig
- Erstellung der Tabellen geschieht durch das Backend

In dieser Vorlesung kommt PostgreSQL zum Einsatz





Backend

- Definiert Datenstruktur mit Tabellen und Entities
- Bevölkert Datenbank mit Daten und liest Daten aus
- Enthält Business Logik der Anwendung
- Enthält REST Endpunkte mit Mappings für Http Funktionen

In dieser Vorlesung kommt Spring mit Kotlin zum Einsatz



Backend



Backend - Bestandteile

Repositories

Lesen der und
Schreiben auf die
Datenbank

Services

*mit DTOs und
Mappern*

Verarbeitung von
Daten und
allgemeine Logik der
Anwendung


Controller

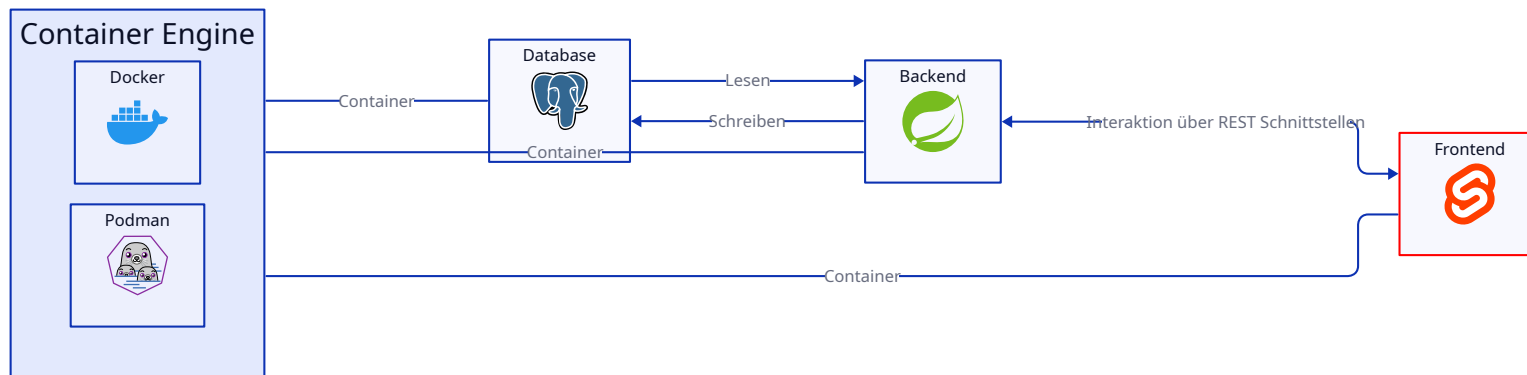
Erstellung der API
Endpunkte über
Mappings

Backend - Gradle

- Automatisierung von Building, Testing und Deployment
 - Verwaltet Abhängigkeiten
 - Reproduzierbare Builds
 - Erweiterbar durch Plugins
-
- Build Vorgang wird in der `build.gradle(.kts)` beschrieben

Backend - Gradle Projekt Struktur

1	project	 Markdown
2	├─ gradle	
3	├─ gradlew	
4	├─ gradlew.bat	
5	├─ settings.gradle(.kts)	
6	├─ subproject-a	
7	└─ build.gradle(.kts)	
8	└─ src/	
9	└─ subproject-b	
10	└─ build.gradle(.kts)	
11	└─ src/	



Frontend

- Interaktion durch den Nutzer
- Darstellung der vom Backend erhaltenen Daten
- Input für neue Daten

Frontend - Bestandteile

- Framework oder Vanilla
- CSS
- Build Tool
- Package Manager
- Weitere Dependencies für unterschiedliche Funktionalitäten

Frontend - Frameworks

In dieser Vorlesung kommt Svelte mit TypeScript zum Einsatz. Andere Frameworks werden auch kurz beleuchtet.



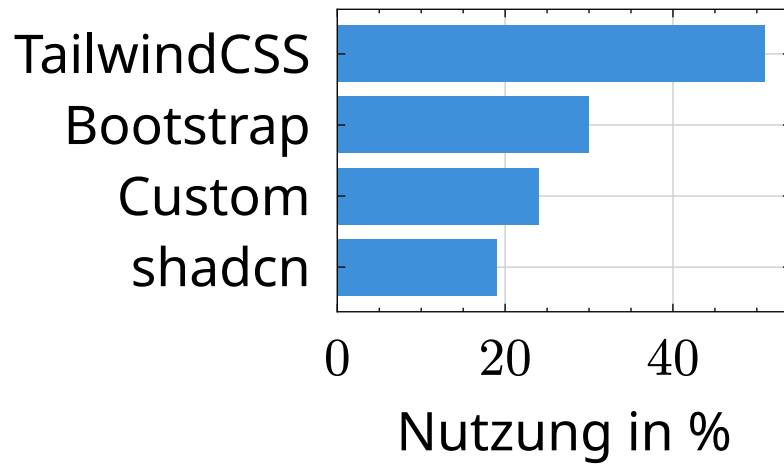
Frontend - Frameworks



- React
- Vue
- Angular

Frontend - Styling

Die aktuelle Verteilung der CSS Frameworks nach dem State of CSS 2025:



Frontend - Styling

- Tailwind CSS als Framework für das Styling
- Es stellt keine Komponenten sondern nur kleine Klassen bereit
- Näher an Vanilla CSS als Frameworks wie Bootstrap

```
1 <div class="flex flex-col bg-slate-500">  
2   <!-- Content -->  
3 </div>
```



Frontend - Build Tools

- Überführen den geschriebenen Code in eine ausführbare Version für den Browser
- Viele Features von Frameworks werden nicht von Browsern unterstützt
- Build Tools überführen diesen Code in brauchbaren Code für den Browser

Frontend - Build Tools

```
1  const App = () => <h1>Hello, World!</h1>;
```

jsx

wird zu

```
1  const App = () => React.createElement(  
2    'h1', null, 'Hello, World!'  
3  );
```

Js JavaScript

Frontend - Build Tools

Bundling

- Zusammenführen von einzelnen Modulen für effizienteres Ausführen im Browser

Transpilieren

- Kompatibilität zu älteren Browsern herstellen

Frontend - Build Tools

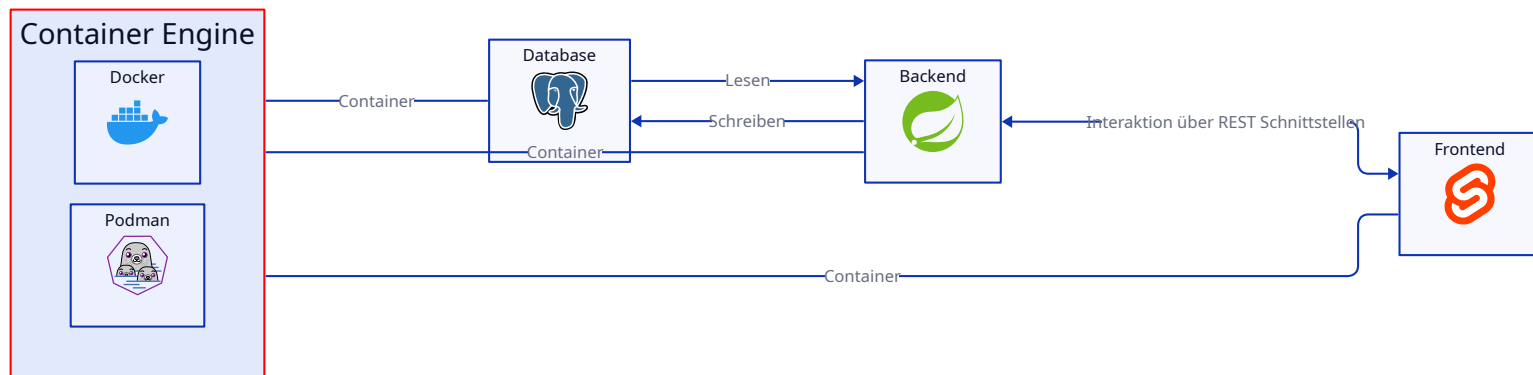


Figure 1: Vite

Frontend - Package Manager

- Definierung der Abhängigkeiten durch eine JSON Datei
- Verwaltung der Abhängigkeiten mit Versionierung
- NPM Registry stellt Packages zum bereit (> 3.100.000)
- Auswahl: npm, yarn, bun, deno usw.
- Alle können die npm registry nutzen aber bieten unterschiedliche Vorteile in der Entwicklungserfahrung
- Vorteile bei npm Alternativen liegen oft in der Performance

It works on my machine



Containerization

- Erstellen von Umgebungen (Containern), die sich, unabhängig von ihrer Umgebung, immer gleich verhalten
- Ähnlich wie eine Virtuelle Maschine, allerdings ohne virtualisierung der Hardware
- Alle Container laufen über den gleichen Kernel -> Mehr Performance als mit VMs
- Software wird in diesen Containern deployed

Containerization - Bestandteile

Dockerfile

Beschreibt den
Aufbau von einem
Image

Image

Eine Abbildung der
Software, die
eingesetzt werden
soll mit allen
benötigten
Abhängigkeiten.

Container

Die laufende
Software,
basierend auf einem
Image

Containerization Tools

Zwei Optionen für Containerization:

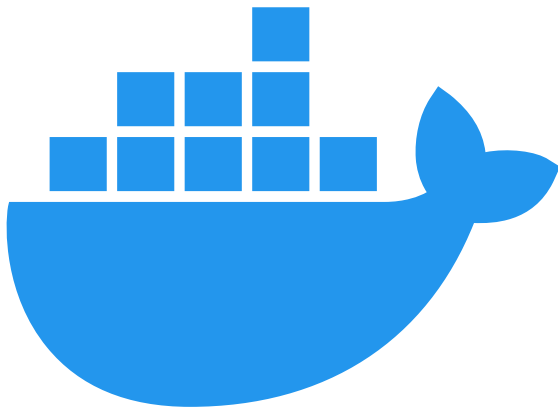


Figure 2: [Docker](https://www.docker.com/)

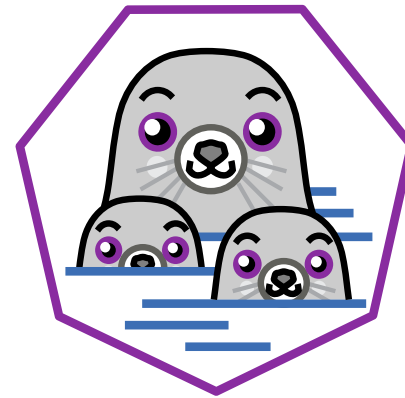


Figure 3: [Podman](https://podman.io/)

Versionierung mit Git

- Versionierung aller Dateien in einem Projekt
- Mit jedem Commit wird ein Snapshot vom aktuellen Stand gemacht
- Wenn keine Änderungen an einer Datei vorgenommen wurden wird auf den Snapshot mit der letzten Änderung verwiesen

Git Zustände

Modified

- Datei wurde geändert aber noch in die Datenbank committed

Staged

- Modifizierte Datei wurde markiert, damit sie in den nächsten Commit Snapshot kommt

Committed

- Daten wurde erfolgreich in der Datenbank gespeichert

Git Workflow

1. Modifizierung von Dateien
2. Nur Änderungen stagen, die Teil des nächsten Commits werden sollen
3. Ausführen eines Commits, welcher alle Änderungen in die Datenbank übernimmt, die als staged markiert wurden

Git - Online

GitHub

GitLab von der TU Chemnitz: <https://gitlab.hrz.tu-chemnitz.de/>