Tools, Softwarearchitektur und Softwaredesign des Projektes Noodle

Datum	22. Februar 2022
Ziel des Dokumentes	Beschreibung Softwarearchitektur und Softwaredesign
Autoren	David Brand Dirk Kremer Hannes Rosenthal Johannes Lehm Max Hampel

Änderungshistorie

Datum der Änderung	Bearbeiter	Art der Änderungen
22.02.2022	David Brand, Dirk Kremer, Hannes Rosenthal, Johannes Lehm, Max Hampel	Erstellung des Dokumentes und erste Bearbeitung der einzelnen Bereiche
23.02.2022	David Brand, Dirk Kremer, Hannes Rosenthal, Johannes Lehm, Max Hampel	Detaillierteres Füllen aller Bereiche, Hinzufügen von Diagrammen zur Beschreibung der Architektur

Tools

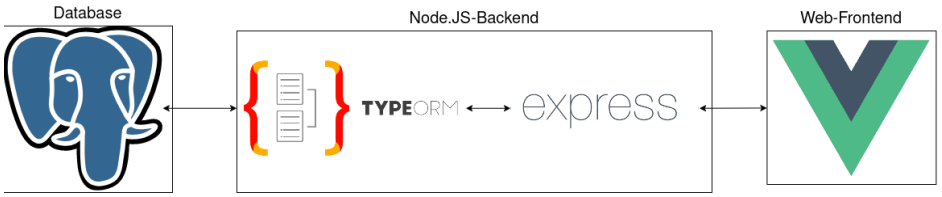
Um Noodle erfolgreich und möglichst effizient zu entwickeln, haben wir uns gemeinsam überlegt, welche Entwicklungstools wir einsetzten möchten. Der Leitgedanke hinter der Auswahl ist, dass wir, insofern möglich, auf Tools zurückgreifen wollen, die wir bereits einsetzen, bzw. in der Vergangenheit eingesetzt haben, um so bereits gewonnen Erfahrung nutzen zu können und die Einarbeitungszeit so gut wie möglich zu reduzieren. So möchten wir erreichen, dass wir möglichst viel Zeit für die tatsächliche Entwicklung von Noodle zur Verfügung haben, um so eine möglichst geringe Time to Market und eine möglichst große Feature List gewährleisten können.
Daher haben wir uns für die folgenden Tools entschieden:

Tools	**Einsatzzweck**
neovim	IDE für die Entwicklung
WebStorm	IDE für Frontend-Entwicklung
VS Code	IDE für Entwicklung
ESLint	- Formatierung Dateien anhand Code Guidelines - Einhaltung Code Guidelines
Github	- Versionskontrolle - Taskmanagement - Dokumentation(mit Hilfe der Readmes & Github Wiki) - Continuous Delivery des Docker Containers mit Hilfe von Github Actions - Continuous Integration des Backends mit Hilfe von Github Actions
Git	Verwaltung von Source Code
Docker-Container	Deployment der Anwendung
npm	Verwaltung von Libraries & Frameworks für Frontend und Backend
Postman, RESTer, Advanced REST Client	Überprüfung Funktionalität API-Schnittstelle
Vue-CLI	Erstellen Vue.js Projekt
Web Browser - Chrome - Firefox - Safari	Debugging

GoodNotes	Frontend-Design
psql	- Anlegen der Datenbank - Auslesen von Daten in der Datenbank - Änderung an Daten in der Datenbank
Jest	- Unit-Tests Backend & Frontend - E2E-Tests Backends
Cypress	- E2E-Tests Frontend

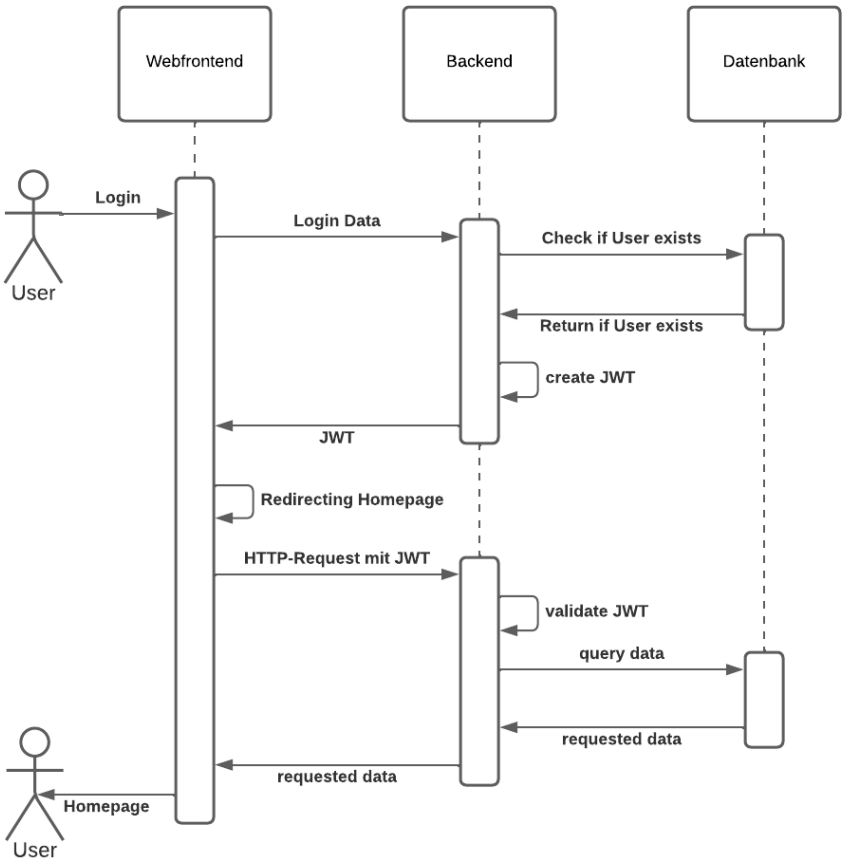
Architektur

Software-Architektur



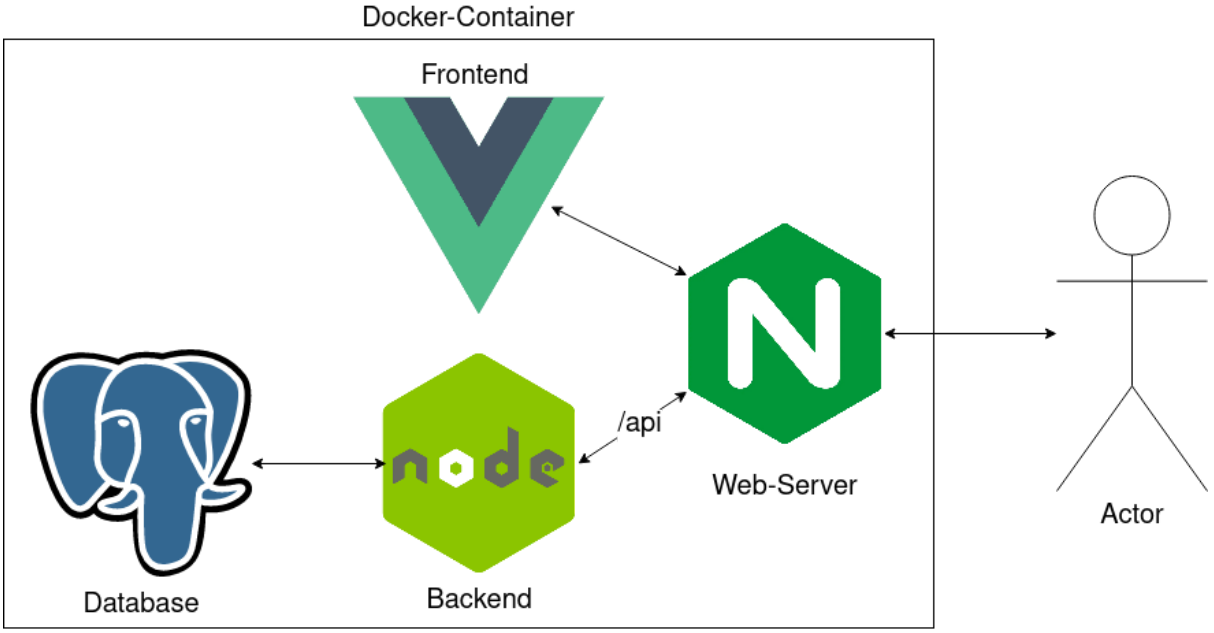
- REST-Architektur
  - Bestehend aus 3 Komponenten
    - PostgreSQL-Datenbank
    - Node.JS-Backend
    - Vue.JS-Frontend
  - Kommunikation zwischen Frontend und Backend über HTTP-Requests
    - Vorgegeben durch Aufruf der Applikation im Browser
  - Kommunikation zwischen Backend und Datenbank über TypeORM-Framework
    - Weniger Fehleranfällig
    - Keine SQL-Statements nötig
    - Kommunikation über gewohnte Methodenaufrufe
  - Authentifizierung über JWT
    - Clientseitig gespeichert
    - Wird bei jedem HTTP-Request mit gesendet

Verhalten des Systems für erfolgreichen Login-Prozess



Deployment-Architektur

- Deployment System in einem Docker-Container
  - Automatisiert gebaut durch Github Actions
  - Verwendung Alpine Linux Container als Basis
  - Verwendung NGINX als zentrale Schnittstelle
    - Bereitstellung Frontend
    - Weiterleitung API-Anfragen an Backend
  - Ermöglicht für den Endbenutzer eine einfache Inbetriebnahme des Systems, da keine Installation und Konfiguration der einzelnen Komponenten benötigt wird



Backend

Verwendetes Software-Design

- Adapter-Pattern
    - Backend nimmt HTTP-Anfragen an und verarbeitet diese mit entsprechender Funktion
    - Erzeugt eine klare Zuordnung der Schnittstelle zur Funktionalität
    - Benötigt, damit mittel des Express-Frameworks Anfragen verarbeitet werden können
- Dekoratoren
    - “Übersetzung” von Attributen einer Klasse in die Attribute einer Datenbanktabelle
    - Klare Zuordnung von Attribute einer Klasse zu Attribute einer Datenbanktabelle
    - Zwingend notwendig für die Verwendung von TypeORM
- Extensible Design
    - Zur Erstellung neuer Tabellen in der Datenbank ist nur die Definierung einer neuen Klasse nötig
- Modulares Design
    - Die Funktionalität der Schnittstelle ist modular unterteilt
        - z.B. /user benutzt nur die Funktionen des Moduls ‘User’
    - Gemeinsames Modul für Interaktion mit der Datenbank
    - Aufteilung der Funktionalität klar strukturiert
    - Einfache Zuordnung eines Pfades zur dahinter liegenden Funktionalität

Frontend

Verwendetes Software-Design

- Composite Pattern
    - Baumstruktur aus einzelnen Komponenten
- Singleton Pattern
    - Nutzung des Vuex-Stores als Singleton-Quelle der States der Komponenten
- State Management Pattern
    - Vuex als zentraler Store, um sicherzustellen, dass States der Komponenten nur in vorhersehbarer Weise verändert werden
- Facade Pattern
    - Abstraktion der Persistenzschicht durch Backend
    - Abstraktion der Datenschicht durch Vuex-Store
- Observer Pattern
    - Vue.js internal data binding
- Dekoratoren
    - z.B. Mappen von Vuex-State auf Variablen einer Komponente
- Extensible Design
    - Verwendung verschiedener Frameworks (Vuetify, Vuex, vue-mq)

Aufgrund von Best Practices zur Datenverwaltung, einer einheitlichen Gestaltung des Frontends und eines angestrebten Responsive Designs haben wir uns für die Verwendung der verschiedenen Frameworks entschieden.
Die verwendeten Design-Pattern sind durch die Architektur von Vue.js und weiterer eingesetzter Frameworks vorgegeben.