Architekt – Abgabe

Softwarearchitektur-Dokumentation

Grundlegendes

Prinzipien

Beschreibung:

- *Einheitlichkeit*: Wir folgen dem Prinzip der Einheitlichkeit in der Codebasis, indem wir konsistente Namenskonventionen und Codierungsstile verwenden.
- *Abstraktion*: Durch die Verwendung von Abstraktionen in Form von React-Komponenten und separaten Modulen verbessern wir die Wartbarkeit und Lesbarkeit des Codes.

Methoden

Beschreibung:

- *Agile Entwicklungsmethodik*: Wir verwenden agile Entwicklungsmethoden wie Scrum, um iterative und inkrementelle Entwicklungszyklen durchzuführen und flexibel auf Änderungen reagieren zu können.
- *Testgetriebene Entwicklung (TDD)*: Bei der Entwicklung neuer Funktionen wenden wir die Testgetriebene Entwicklung an, um die Qualität des Codes zu verbessern und Fehler frühzeitig zu erkennen.
- *Wireflows und Trello-Sprints*: Zur Planung und Organisation unserer Entwicklungsarbeiten verwenden wir Wireflows zur Vordefinition der Benutzeroberfläche und Trello für die Aufteilung der Arbeitspakete in Sprints. Dabei haben wir in Trello die Spalten "New", "Active" und "Sprint 1 Closed" bis "Sprint n Closed", um den Fortschritt und die Abwicklung unserer Arbeitspakete transparent zu verfolgen.

Werkzeuge

Auflistung:

- *Next.js*: Ein React-Framework für die Entwicklung von Webanwendungen, das Server-Side Rendering und einfache API-Integration ermöglicht.

- *Prisma ORM*: Ein modernes Datenbank-Toolkit für TypeScript und Node.js, das die Interaktion mit der Datenbank erleichtert.
- *SQLite*: Eine leichte relationale Datenbank, die lokal in der Anwendung eingebettet ist und für Entwicklungszwecke verwendet wird.
- *Trello*: Wir haben das Trello-Board genutzt, um unsere Arbeitspakete aufzuteilen.

Techniken

Beschreibung:

Wir setzen auf TypeScript für die statische Typisierung und bessere Codequalität. React wird als Frontend-Bibliothek verwendet, um eine komponentenbasierte Architektur zu ermöglichen. Für die Verwaltung des Zustands verwenden wir das Context-API von React zusammen mit Hooks.

Alternativen

Erläuterung:

Bei der Auswahl der Technologien wurden verschiedene Alternativen in Betracht gezogen. Als Alternative zu unserer aktuellen Auswahl wurden Java/C# und Python als Backend eine Überlegung Wert, da jeder von uns jeweils nur in einem der Backend Frameworks vertieft war. Letztendlich wurden die ausgewählten Technologien aufgrund ihrer Eignung für das Projekt und der Teamabstimmung gewählt. Dabei wurde die Wahl der Person mit der Entwickler Rolle am stärksten gewichtet.

Modularisierung

Frameworks, Patterns

Für die Modularisierung unseres Projekts verwenden wir verschiedene Frameworks und Bibliotheken, um die Entwicklung zu erleichtern und bewährte Praktiken umzusetzen. Einige der Hauptframeworks und Bibliotheken sind:

- *Next.js*: Next.js wird als React-Framework für das Server-Side Rendering (SSR) und das Static Site Generation (SSG) verwendet. Es bietet eine intuitive und leistungsstarke Möglichkeit, React-Anwendungen zu erstellen und zu verwalten.
- *Prisma ORM*: Prisma ORM wird für die Datenbankinteraktion und das Datenbankmanagement verwendet. Es erleichtert das Arbeiten mit Datenbanken und bietet eine sichere und robuste Methode zum Ausführen von Datenbankabfragen.
- *SQLite*: SQLite wird als relationale Datenbank für die lokale Entwicklung und Tests verwendet. Es ist eine leichte, aber leistungsstarke Datenbanklösung, die einfach zu integrieren ist und eine gute Leistung bietet.

- *Typescript*: Typescript wird als Programmiersprache verwendet, um die Typsicherheit in unserer Anwendung zu verbessern und die Entwicklungszeit zu verkürzen. Es ermöglicht eine klarere und strukturiertere Codebasis und hilft, Fehler frühzeitig zu erkennen und zu vermeiden.
- *React*: React wird als Hauptbibliothek für die Benutzeroberfläche verwendet. Es bietet eine effiziente und deklarative Möglichkeit, interaktive UI-Komponenten zu erstellen und zu verwalten.

Packages/Verzeichnisse

Unser Projektverzeichnis ist gut strukturiert und folgt bewährten Praktiken für die Organisation von Code und Ressourcen. Einige der wichtigsten Verzeichnisse sind:

- *prisma*: Enthält die Datenbankmigrationen und Konfigurationen für Prisma ORM.
- *migrations*: Enthält die durchgeführten Datenbankmigrationen zur Verwaltung des Datenbankschemas.
- *public*: Enthält öffentlich zugängliche Dateien und Ressourcen, die direkt vom Client ausgeliefert werden.
- *src*: Enthält den Quellcode unserer Anwendung.
- *app*: Enthält den Hauptcode unserer Anwendung, einschließlich der Hauptkomponenten und Logik.
- *actions*: Enthält Funktionen zur Interaktion mit der Datenbank und anderen externen Diensten.
- *admin*: Enthält Komponenten und Logik für die Verwaltung von Benutzern, Buchungen und anderen administrativen Aufgaben.
- *api*: Enthält API-Endpunkte und Logik für die Authentifizierung und Autorisierung von Benutzern.
- *booking*: Enthält Komponenten und Logik für die Buchung von Ständen und Vorträgen.
- *bookings*: Enthält Komponenten und Logik für die Anzeige und Verwaltung von Buchungen.
- *components*: Enthält wiederverwendbare UI-Komponenten, die in der gesamten Anwendung verwendet werden.
- *dashboard*: Enthält Komponenten und Logik für das Dashboard und die Übersicht.
- *datenschutz*: Enthält Komponenten und Inhalte für die Datenschutzrichtlinie der Anwendung.

- *home*: Enthält Komponenten und Logik für die Startseite der Anwendung.
- *infos*: Enthält Informationen und Dokumentationen zur Anwendung.
- *login*: Enthält Komponenten und Logik für die Anmeldung und Authentifizierung von Benutzern.
- *logout*: Enthält Logik für die Abmeldung von Benutzern.
- *profile*: Enthält Komponenten und Logik für das Benutzerprofil und die Einstellungen.
- *register*: Enthält Komponenten und Logik für die Registrierung neuer Benutzer.
- *store*: Enthält Zustandsmanagement mit Zustand (Zustand) und Reaktion (React).

Visualisierung

Siehe Anhang im Ordner.

Typische Dokumenteninhalte

Siehe Anhang im Ordner.