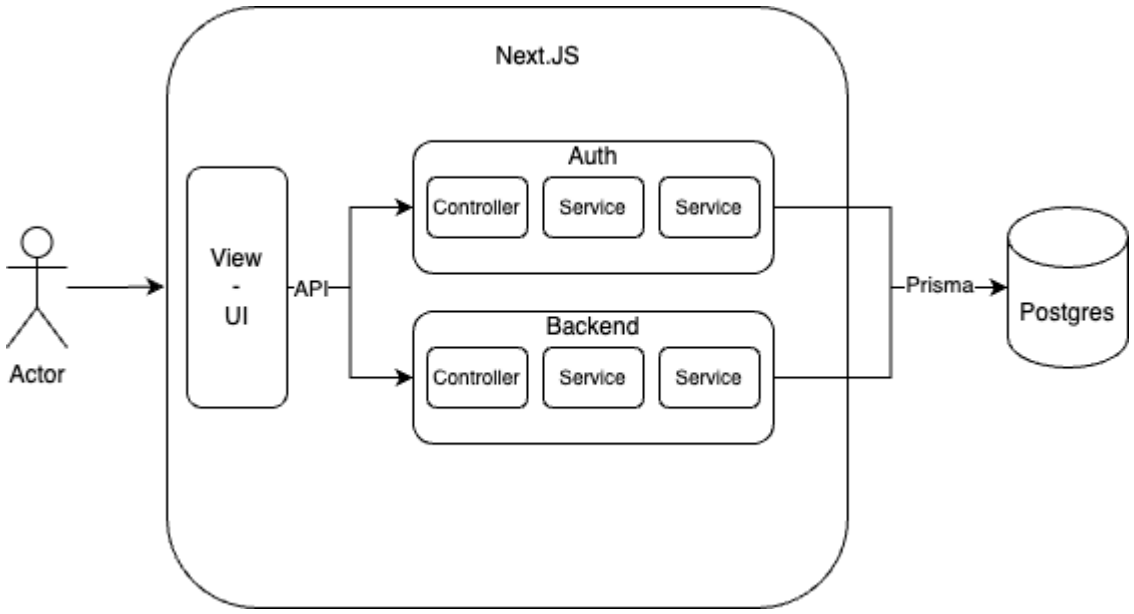
INF202 - Meilenstein #3

Seda Deniz ÖZDOĞAR e180502006@stud.tau.edu.tr
Gülşah TİYEK e170502003@stud.tau.edu.tr

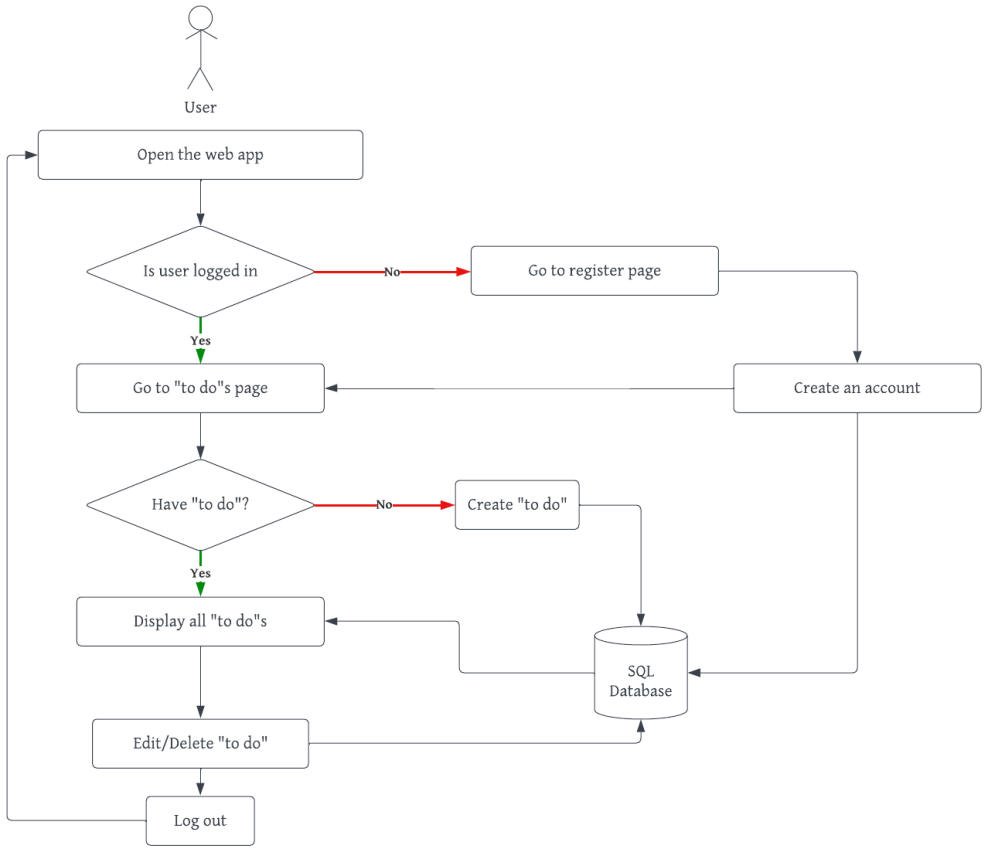
Chapter 1

Wir haben unser Projekt mit Next.JS entwickelt, einem JavaScript-basierten Fullstack-Framework. Dank Next.JS ermöglichen sowohl View (Frontend) als auch Controller (Backend) die Entwicklung innerhalb desselben Projekts. Wir haben unsere Anwendung mit React entwickelt, einem weiteren Javascript Frontend-Framework im View-Layer. Dank React lassen sich Schnittstellen einfach gestalten. Benutzer können die in den Geschäftsanforderungen definierten Aktionen über die Schnittstelle ausführen, die wir mit React.JS entwickelt haben. Diese Aktionen werden mit der RESTful API an das Backend übermittelt. Die an das Backend kommende Anfrage wird zunächst in der Controller-Schicht validiert und gelangt dann gemäß der Geschäftslogik in die Service-Schicht.

Danach werden die Daten zur Speicherung in die Datenbank übertragen. Vor der Übertragung auf die Datenbank wird ein Repository namens Prisma verwendet, um eine komfortable Backend-Datenbank-Beziehung herzustellen. Das allgemeine Projekt Diagramm ist wie unten gezeigt. Auth wurde nicht als externer Dienst hinzugefügt, da es noch nicht abgeschlossen ist.



UI Use Cases



1. Registrieren: Der Benutzer soll sich registrieren können, wenn kein Konto vorhanden ist.
2. Login: Der Benutzer sollte in der Lage sein, sich in das nach der Registrierung erstellte Konto einzuloggen.
3. Aufgaben anzeigen: Der Benutzer soll seine Aufgaben anzeigen können.
4. Aufgaben hinzufügen: Der Benutzer sollte in der Lage sein, der Aufgabenliste neue Aufgaben hinzuzufügen, indem er sie in ein Textfeld eingibt und auf eine Schaltfläche klickt. Die App sollte die neue Aufgabe zusammen mit allen zuvor hinzugefügten Aufgaben in der Liste anzeigen.
5. Aufgaben bearbeiten: Der Benutzer sollte in der Lage sein, die Beschreibung einer Aufgabe in der Liste zu bearbeiten, indem er darauf klickt und den neuen Text eingibt. Die App sollte die Änderungen speichern und die Liste entsprechend aktualisieren.
6. Aufgaben als abgeschlossen markieren: Der Benutzer sollte in der Lage sein, eine Aufgabe als abgeschlossen zu markieren, indem er ein Kästchen daneben in der Liste aktiviert. Erledigte Aufgaben sollte die App optisch von unvollständigen unterscheiden, etwa durch Ausgrauen.
7. Aufgaben löschen: Der Benutzer sollte in der Lage sein, eine Aufgabe aus der Liste zu löschen, indem er auf eine Schaltfläche zum Löschen daneben klickt. Die App sollte die Aufgabe aus der Liste entfernen und die Anzeige entsprechend aktualisieren.

API Use Cases

1. Authentifizierung und Autorisierung: Die API sollte sichere Authentifizierungs- und Autorisierungsmechanismen bereitstellen, um sicherzustellen, dass nur autorisierte Benutzer auf ihre Aufgabenlisten zugreifen und diese ändern können.
2. Aufgaben erstellen: Die API sollte es Benutzern ermöglichen, neue Aufgaben zu erstellen, indem sie eine POST-Anforderung mit einer Aufgabenbeschreibung, Fälligkeitsdatum, Priorität und anderen relevanten Informationen senden. Die API sollte eine Antwort mit der ID der neu erstellten Aufgabe und anderen Details zurückgeben.
3. Aufgaben abrufen: Die API sollte es Benutzern ermöglichen, ihre Aufgabenlisten abzurufen, indem sie eine GET-Anforderung mit ihren Authentifizierungsdaten senden. Die API sollte eine Antwort mit einer Liste aller Aufgaben des Benutzers und deren Details zurückgeben, z. B. Aufgabenbeschreibung, Fälligkeitsdatum, Priorität und Abschlussstatus.
4. Aufgaben aktualisieren: Die API sollte es Benutzern ermöglichen, vorhandene Aufgaben zu aktualisieren, indem sie eine PUT- oder PATCH-Anforderung mit der Aufgaben-ID und allen aktualisierten Informationen senden. Die API sollte eine Antwort mit den aktualisierten Aufgabendetails zurückgeben.
5. Aufgaben als abgeschlossen markieren: Die API sollte es Benutzern ermöglichen, eine Aufgabe als abgeschlossen zu markieren, indem sie eine PUT- oder PATCH-Anfrage mit der Aufgaben-ID und dem Abschlussstatus senden. Die API sollte eine Antwort mit den aktualisierten Aufgabendetails zurückgeben.
6. Aufgaben löschen: Die API sollte es Benutzern ermöglichen, eine Aufgabe zu löschen, indem sie eine DELETE-Anforderung mit der Aufgaben-ID senden. Die API sollte eine Antwort zurückgeben, die bestätigt, dass die Aufgabe gelöscht wurde.

Chapter 2

Im Zusammenhang mit einer To-do-Anwendung ist die Controller-Schicht typischerweise der Teil der Anwendung, der für die Bearbeitung von Benutzeranforderungen und die entsprechende Aktualisierung der Daten und der Benutzerschnittstelle der Anwendung verantwortlich ist.

Wenn ein Benutzer beispielsweise über die Benutzeroberfläche der Anwendung ein neues Element zu seiner Aufgabenliste hinzufügt, erhält die Controller-Schicht diese Anfrage und aktualisiert das Datenmodell entsprechend, um sicherzustellen, dass das neue Element gespeichert und ordnungsgemäß in der Benutzeroberfläche angezeigt wird.

In unserem Fall können nur eingeloggte Benutzer Aufgaben mit maximal 300 Zeichen erstellen. Die Controller-Schicht prüft also zuerst, ob der Benutzer angemeldet ist, kontrolliert dann die Höchstgrenzen und sendet die Aufgaben an die Dienstschicht.

Wir haben 3 verschiedene Main RESTful API.

API Endpoints

- localhost:3000/login
- localhost:3000/signup
- localhost:3000/todos

API Methods

- POST: localhost:3000/login
    - Required Fields: Email, Password
- POST: localhost:3000/signup
    - Required Fields: Email
- GET: localhost:3000/todos
    - Get All Todos
- POST: localhost:3000/todos
    - Required Fields: Todo, resolution, account
- PATCH: localhost:3000/todos/[id]
    - To change resolution of todo
- DELETE: localhost:3000/todos/[id]
    - To delete todo
    - Only users with admin role true can use this endpoint

Chapter 3

- Controller-Layer überprüft Benutzer, die angemeldet sind oder nicht
- Die Überprüfung der Controller-Layer ist todo für unser Datenmodell geeignet, das maximal 300 todo-Zeichen akzeptiert
- Controller todo sendet neue Todos an den Dienst, da die Lösung nicht abgeschlossen ist
- Wenn der Benutzer die ToDo-Auflösung ändert, prüft der Controller, ob Enum für die Datenschicht geeignet ist
- Benutzer der Controller-Layer können nur ihre eigenen Aufgaben löschen
- Der Administrator der Controller-Layer kann alle Daten sehen und löschen

Chapter 4

Postgres, eine relationale Datenbank, wird bevorzugt. Seine Datenbank arbeitet auf Railway als serverlos. Skalierbarkeit und Leistung werden dank der serverlosen Architektur verbessert. Next.JS unterstützt ORM mit dem Namen PRISMA an sich. ORMs wie Prisma helfen dabei, Datenbankabfragen zu schreiben und Daten einfach zu manipulieren. Prisma arbeitet in der Repository-Schicht, die eine Verbindung zwischen Datenbank und Backend-Anwendung herstellt.