TAU INF202 Software Engineering

Individuelles Projekt

Pflichtenheft

Projektdokumentation

Version: 2023.v1.0

Status: freigegeben

To Do App

Verantwortliche/r:

Seda Deniz Özdoğar, e180502006@stud.tau.edu.tr

Gülşah Tiyek, e170502003@stud.tau.edu.tr

Berater: Çağatay Yılmaz, mail@hcagatay.com

Stakeholder: DI. Ömer Karacan, omer.karacan@tau.edu.tr

Template: Projektdokumentation.doc / Version vom 29.09.2020

Dokumentenverwaltung

Dokument-Historie

Version	Status *)	Datum	Verantwortlicher	Änderungsgrund
v1.0	freigegeben	17.04.2023	Ö. Karacan	Erste Version fürs Projekt

*) Sofern im Projekt nicht anders vereinbart, sind folgende Statusbezeichnungen zu verwenden (in obiger Tabelle und am Deckblatt):
Dokument-Status: Entwurf / in Review / freigegeben (abgegeben)

Dokument wurde mit folgenden Tools erstellt:
Microsoft Office Word
diagrams.net, excalidraw.com, draw.io

Inhaltsverzeichnis

1. Einleitung

Dieses Dokument dient dazu, die Anforderungen an eine „To Do App“ zu definieren und diese Anforderungen vollständig und schlüssig zu erläutern.

Anwendungsfälle und Anforderungen werden aus Sicht der Stakeholder definiert.

Die grafische Oberfläche für Überwachungsaufgaben wird aus der Sicht des Benutzers (meistens) und Administrators erklärt.

Kapitel 2 „Ausgangslage und Ziele“ zeigt anschaulich die Ausgangslage und den Grund für die Wahl der “To Do App“.

Im Kapitel 3 „Gesamtarchitektur“ sind die physikalische und die konzeptionelle Architektur des Systems, und die wichtigsten Subsysteme (Komponenten), die Anwender und die notwendigen Kommunikationsschnittstellen dargestellt. Hier sind auch zusätzliche Anforderungen an die Architektur oder Komponenten zu finden.

Im Kapitel 4 „Funktionale Anforderungen“ beinhaltet die Beschreibung der funktionalen Anforderungen durch die Ablaufbeschreibungen (User Stories), die Anwendungsfälle (Use Cases) und technische und fachliche Anforderungen (Requirements). Alle betriebsrelevanten Daten werden durch die Datenmodelle definiert.

Im Kapitel 5 „Nichtfunktionale Anforderungen“ sind die funktionalen Anforderungen durch spezielle Anforderungen erweitert, die keine funktionalen Anforderungen sind.

Im Kapitel 6 „Abnahmekriterien“ sind die Abgabeartefakten festgelegt, die ohne Abstimmung des Stakeholders nicht zu manipulieren sind.

Im Kapitel 7 „Projekt Meilensteine“ sind die wichtigsten Termine ausgelistet, die den Fortschritt die Teilergebnisse des Projektes definieren.

Im Kapitel 8 „Referenzen“ sind die wichtigsten Referenzen aufgelistet.

2. Ausgangssituation und Ziele

In diesem Kapitel werden die Ausgangssituation und der Grund zur Wahl des Projektthemas anschaulich dargestellt.

Einleitung

Ziel ist es, die Produktivität zu steigern, indem es den Benutzern das Leben erleichtert.

Die Benutzer organisieren die Aufgaben, Ziele und Anforderungen, die mit der Aufgabenverwaltungs-App erledigt werden müssen. Sie können neue Aufgaben hinzufügen, Aufgaben bearbeiten und löschen, indem sie sich mit ihrem Google-Konto bei der Anwendung anmelden. Erledigte Aufgaben können sie als „erledigt“ markieren. Auf diese Weise wird ihr Leben in Ordnung gebracht.

Es gibt auch einen Administrator im System. Der Administrator kann Aufgaben aller Benutzer anzeigen, bearbeiten und löschen. Wenn der Administrator eine Aufgabe hinzufügt, wird die hinzugefügte Aufgabe auf der Startseite angezeigt. Die Benutzer können nur ihre eigenen Aufgaben einsehen, sie sind nicht berechtigt, die Aufgaben anderer Benutzer einzusehen.

Problemstellung (Funktionalität)

Heutzutage wird es immer schwieriger, der zunehmenden Arbeitsbelastung und Verantwortung zu folgen. Daher erklärt dieses Dokument vollständig die "To Do App", die Benutzern hilft, ihre Aufgaben, Ziele und Wünsche zu organisieren.

Stakeholder (Anwender):

Es wurde für Benutzer erstellt, die Schwierigkeiten haben, ihr Leben zu organisieren und ihre Aufgaben / Jobs / Anforderungen regelmäßig verfolgen müssen.

Systemumfeld (Einsatzumgebung)

Cloud hosting, Vercel

Rahmenbedingung (Einschränkungen)

- Language: Javascript, SQL, HTML/CSS
- Framework: Next.js
- IDE: Visual Studio Code
- Database: PostgresQL
- Authentication: NextAuth

Ziele (Lösung)

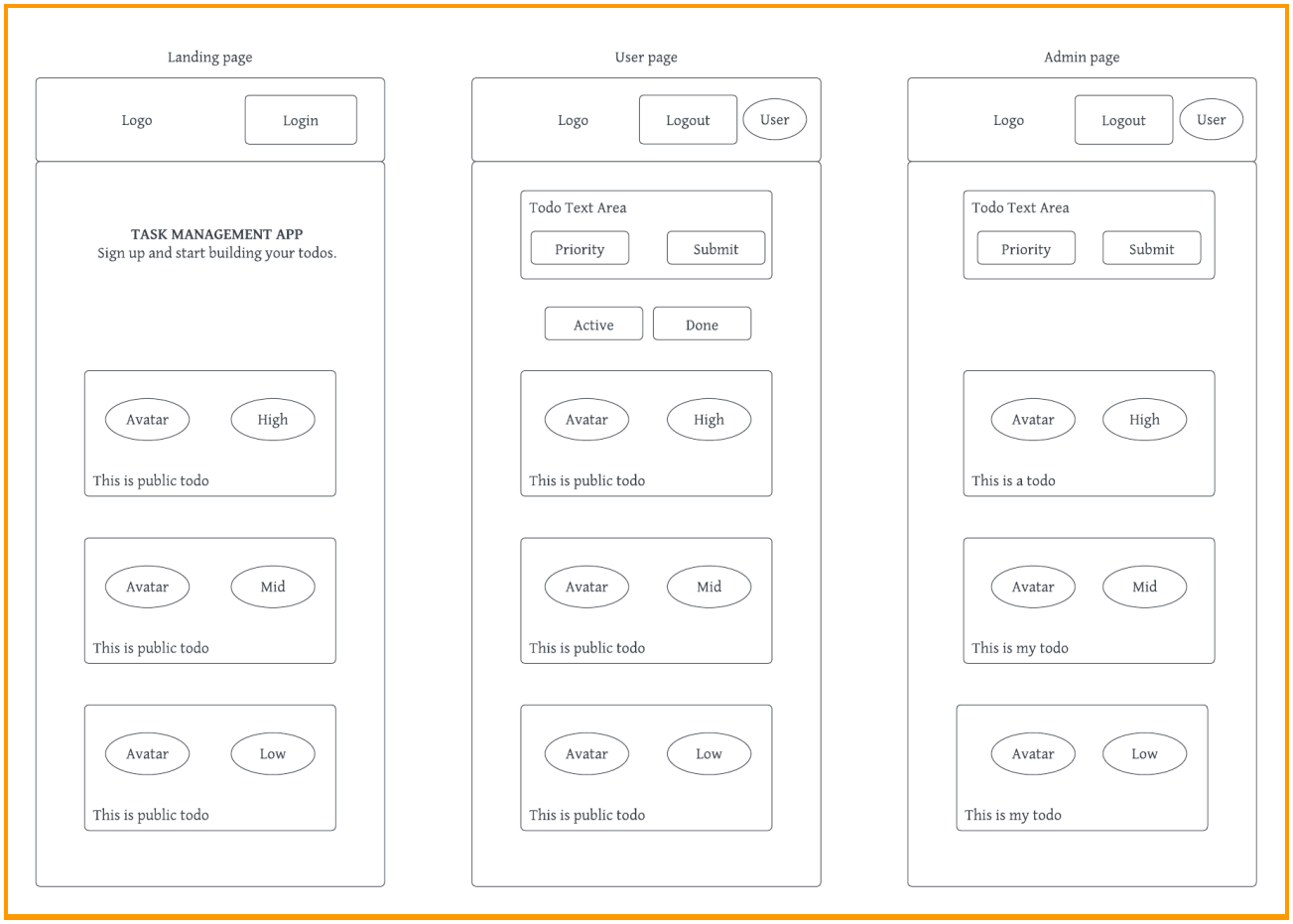
Die meisten to do apps werden bezahlt, es ist schwierig, eine benutzerfreundliche To Do-App zu finden, ohne Geld zu bezahlen. Mit der von uns entwickelten Anwendung bieten wir einen benutzerfreundlichen und kostenlosen Service. Auf diese Weise entwickeln wir eine Anwendung, die für alle zugänglich ist. Jede Anfrage von Benutzern wird über die Schnittstelle zugänglich sein.

3. Gesamtarchitektur

Um die Visualisierung der Anwenderanforderungen zu ermöglichen, soll Gesamtsystemarchitektur illustriert werden, die die Sichtweise des Anwenders repräsentiert und nicht die technische Sichtweise des Systemanalytikers beziehungsweise des Systemarchitekten.

Es soll eine konzeptionelle Architektur unter der Berücksichtigung der Kommunikation (Interaktionen) mit den externen Systemen erstellt werden (Anm.: Vergleich mit einer technischen Systemarchitektur).

Darüber hinaus, in der Gesamtsystemarchitektur sollten die zukünftigen Systembestandteile (Komponente) identifiziert und festgeschrieben werden.



Einleitung

In diesem Kapitel sind die Systemgrenzen definiert. Unter Berücksichtigung der Systemgrenzen sind per Anwender die externen Schnittstellen definiert (graphische Schnittstellen und API). Die notwendigen Systemkomponenten und Datenstrukturen sind definiert und modelliert.

Dieses Kapitel führt zu dem besseren Verständnis des angeforderten Systems und dadurch genaue Definition der funktionalen und nicht funktionalen Anforderungen.

Gesamtarchitektur

Die Gesamtarchitektur von To Do App basiert auf einer Client-Server-Architektur. Die Architektur besteht aus drei Hauptkomponenten:

- Client
- Webserver
- Datenbank

Komponente <x>

Die Client-Komponente ist die Benutzeroberfläche von To Do App mit der Benutzer ihre Aufgaben erstellen, bearbeiten und löschen können. Der Client ist eine Single-Page-Anwendung. Der Benutzer reicht seine Aufgaben ein, indem er Anfragen stellt/erstellt.

Der Webserver ist in Javascript geschrieben und verwendet Next,js. Der Server überprüft die Authentifizierungsinformationen des Benutzers. Es validiert Anfragen des Benutzers und handhabt Middleware. Jede vom Benutzer gestellte Anfrage wird als Restfull an die Server gesendet und von dort wird eine Antwort zurückgegeben. Unangemessene “to do’s“ werden nicht an die Datenbank gesendet. Ermöglicht Benutzern, ihre eigenen Aufgaben zu sehen.

Die Datenbank ist die Komponente, die dauerhafte Daten von „To Do App“ speichert. Es ist eine in SQL geschriebene relationale Datenbank. Die Datenbank enthält Tabellen für Benutzer, Administratoren, Aufgaben und andere zugehörige Daten. Todos werden in der Datenbank gespeichert. Neu eingehende Aufgaben werden in die Datenbank eingefügt, aktualisiert und gelöscht.

Externe Schnittstellen

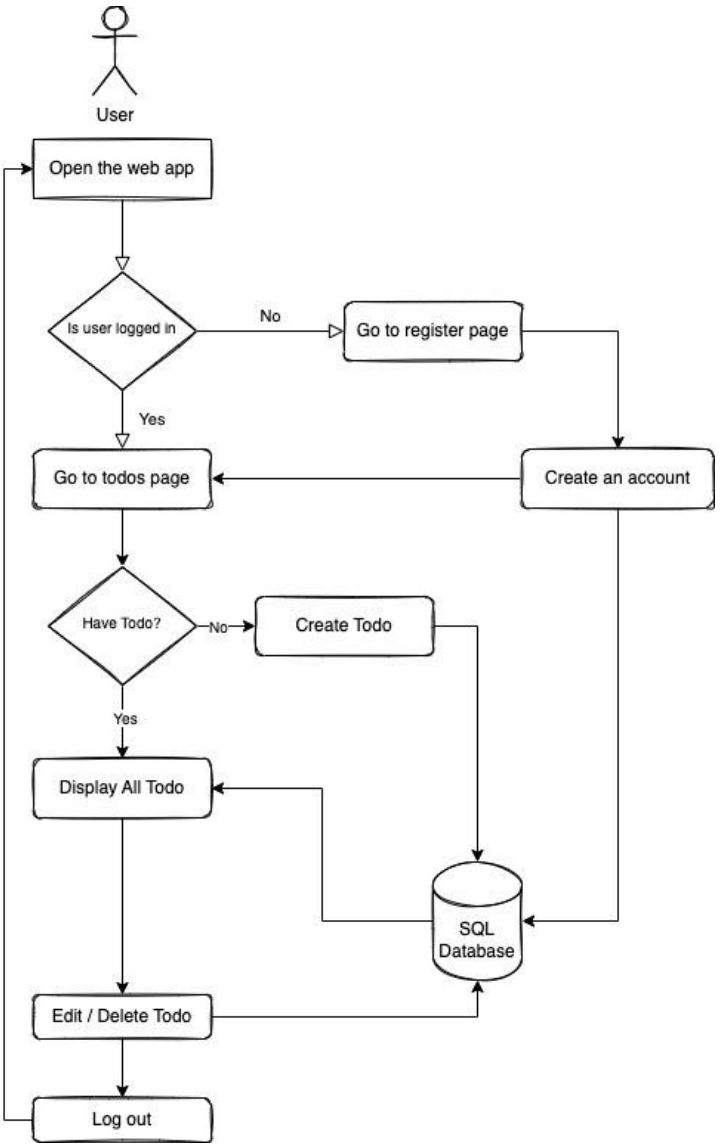
*keine externe Schnitstelle

4. Funktionale Anforderungen

Einleitung

Dieses Kapitel enthält alle UI- und API-Funktionsanforderungen unserer Aufgabenverwaltungsanwendung. Funktionale Anforderungen (FRs) sind eine Art von Softwareanforderungen, die angeben, was das Softwaresystem tun oder welche Funktionalität es seinen Benutzern bieten soll. Diese Anforderungen beschreiben die spezifischen Merkmale, Verhaltensweisen und Fähigkeiten des Systems, die implementiert werden müssen, um die Anforderungen seiner Benutzer zu erfüllen. UI-Anwendungsfälle zeigen Szenarien, die Benutzer über die Schnittstelle ausführen können. API-Anwendungsfälle hingegen demonstrieren den Prozess, wenn Benutzer ihre Anfragen ausführen, bis sie in die Datenbank geschrieben werden.

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.

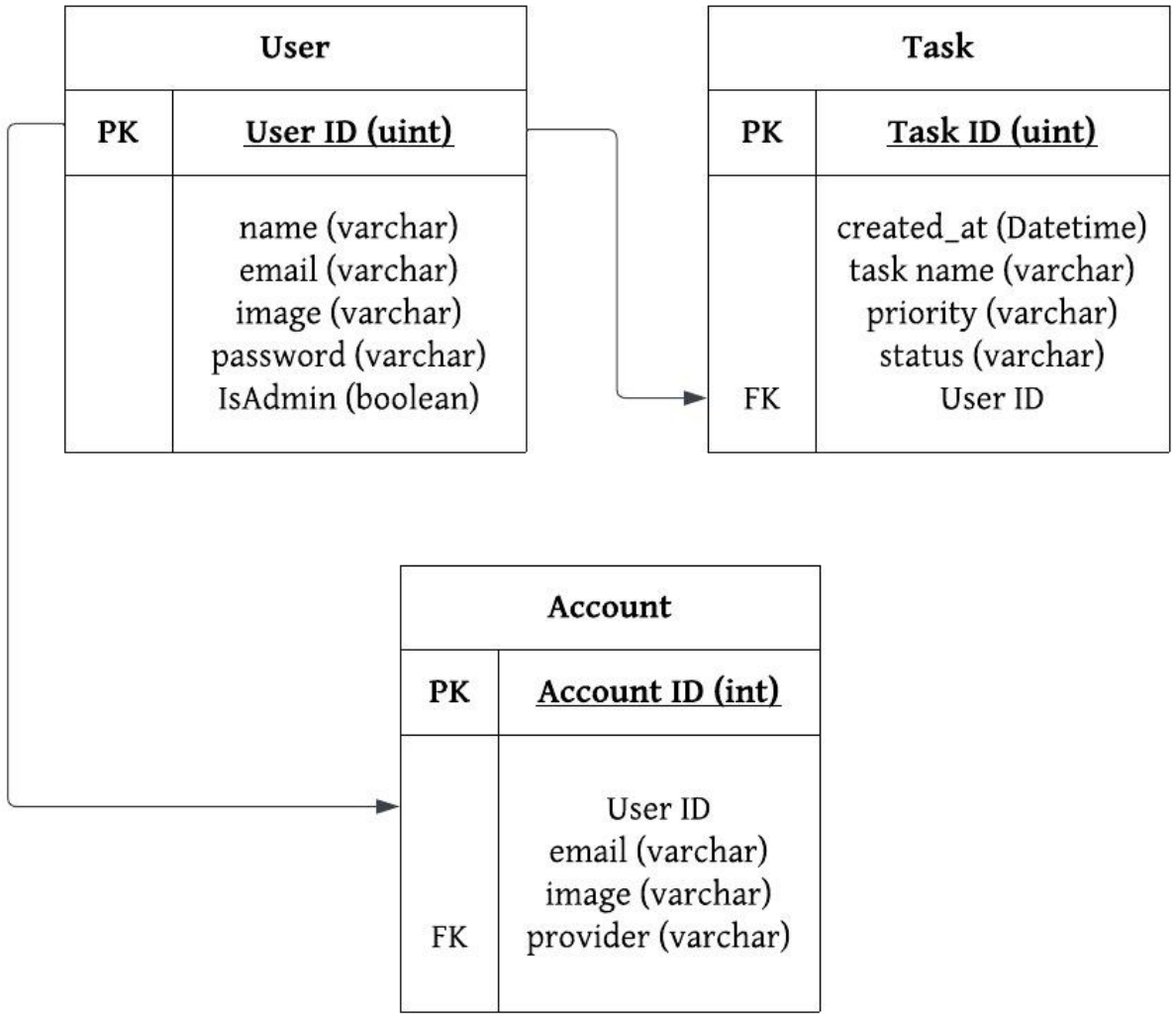
6. Aufgaben löschen: Die API sollte es Benutzern ermöglichen, eine Aufgabe zu löschen, indem sie eine DELETE-Anforderung mit der Aufgaben-ID senden. Die API sollte eine Antwort zurückgeben, die bestätigt, dass die Aufgabe gelöscht wurde.

Technischen und fachliche Anforderungen

1. Wenn ein Fehler auftritt, werden Benutzer mit einer Fehlermeldung benachrichtigt.
2. Der Benutzer wird benachrichtigt, wenn das Kennwort falsch ist.

Datenmodel

Datenbanktabellen sind wie im Diagramm dargestellt und bestehen aus Benutzerkonto- und Aufgabentabellen. Darüber hinaus wird eine separate Sitzung für Sitzungsinformationen verwendet.



4. Nichtfunktionale Anforderungen

Einleitung

Nichtfunktionale Anforderungen (NFR) haben keinen direkten Bezug zur Funktionalität eines Softwaresystems. Sie sind Anforderungen, die Kriterien angeben, die die Merkmale des Systems wie Verfügbarkeit, Zuverlässigkeit, Skalierbarkeit, Sicherheit und Leistung beschreiben.

Nicht-funktionale Anforderungen an die Systemarchitektur (Architekturmuster, Deployment)

1. Skalierbarkeit: Die Anwendung muss so konzipiert und bereitgestellt werden, dass sie als Reaktion auf Änderungen der Benutzernachfrage nach oben oder unten skaliert werden kann.

2. Verfügbarkeit: Die Anwendung muss hochverfügbar und belastbar sein, um sicherzustellen, dass Benutzer jederzeit darauf zugreifen können. Dies kann die Verwendung von Techniken wie redundanter Infrastruktur, Failover-Mechanismen sowie Sicherungs- und Wiederherstellungsverfahren beinhalten.

3. Kompatibilität: Die Anwendung muss mit den neuesten Versionen gängiger Webbrowser kompatibel sein, einschließlich Chrome, Firefox, Safari und Edge.

4. Sicherheit: Die Anwendung muss so konzipiert und bereitgestellt werden, dass die Vertraulichkeit, Integrität und Verfügbarkeit von Benutzerdaten gewährleistet ist. Wir werden Techniken wie Verschlüsselung, Zugriffskontrolle und sichere Codierungspraktiken verwenden, um unbefugten Zugriff oder Datenschutzverletzungen zu verhindern.

5. Wartbarkeit: Die Anwendung muss im Laufe der Zeit einfach zu warten und zu aktualisieren sein, mit einem Minimum an technischer Schuld und einer klaren Trennung von Bedenken zwischen verschiedenen Komponenten. Dies kann den Einsatz von Techniken wie modularem Design, Versionskontrolle und automatisierten Tests umfassen, um sicherzustellen, dass Änderungen schnell und sicher vorgenommen werden können.

6. Leistung: Die Anwendung muss so konzipiert und bereitgestellt werden, dass schnelle Antwortzeiten und geringe Latenz mit einer maximalen Seitenladezeit von 4 Sekunden und einer maximalen API-Antwortzeit von 3 Sekunden gewährleistet sind. Dies kann die Verwendung von Techniken wie Caching, Komprimierung und Optimierung beinhalten, um die Seitenladezeiten zu verkürzen und die Gesamtleistung zu verbessern.

Nicht-funktionale Anforderungen an die Entwicklungsumgebung

1. Leistungsüberwachung: Die Entwicklungsumgebung muss über Leistungsüberwachungstools verfügen, um Metriken wie Antwortzeit, CPU-Auslastung und Speichernutzung zu verfolgen, um Leistungsprobleme früh im Entwicklungsprozess zu erkennen.

2. Kontinuierliche Integration und Bereitstellung: Die Entwicklungsumgebung muss die kontinuierliche Integration und Bereitstellung unterstützen, wobei Tools wie Jenkins oder CircleCI verwendet werden, um den Build- und Bereitstellungsprozess zu automatisieren und sicherzustellen, dass neuer Code schnell und zuverlässig getestet und bereitgestellt wird. Wir planen, Vercel zu verwenden, um unsere Anwendung bereitzustellen.

3. Zusammenarbeit: Die Entwicklungsumgebung muss die Zusammenarbeit zwischen mehreren Entwicklern unterstützen, mit Versionskontrolltools wie Git und einem gemeinsamen Repository zum Speichern von Code und Dokumentation. Wir werden GitHub für die Zusammenarbeit verwenden.

Nicht-funktionale Anforderungen an die Entwicklungswerkzeuge (Sprache, IDE, Frameworks)

Wir werden Javascript als Programmiersprache für unser Projekt verwenden. Javascript ist eine sehr beliebte Webprogrammiersprache und unterstützt auch das Schreiben von Backend-Code mit der Node.js-Laufzeit. Wir planen, Next.js als Full-Stack-Javascript-Framework zu verwenden. Der Hauptgrund für die Wahl von Next.js ist, dass es Ihnen ermöglicht, Frontend- und Backend-Code an derselben Stelle zu schreiben. Es bietet auch eine Hochleistungsinfrastruktur, wenn es in Vercel gehostet wird, das auch das Framework entwickelt hat. Für die integrierte Entwicklungsumgebung (IDE) verwenden wir Visual Studio Code, da es integrierte Github-Unterstützung bietet.

Nicht-funktionale Anforderungen an die Teststrategie (Qualitätssicherung)

Nachdem die Entwicklung abgeschlossen ist, werden alle Plattformen manuell getestet, um sicherzustellen, dass es keine Probleme gibt. Wir werden auch einen Leistungstest planen, um sicherzustellen, dass die Leistung für unsere nicht funktionalen Anforderungen zufriedenstellend ist. Da wir die Verwendung der Google-Authentifizierung planen, führen wir schließlich auch einen Sicherheitstest durch, um jegliche Art von Datenschutzverletzungen zu verhindern.

6. Abnahmekriterien

Die Abnahmekriterien sind durch den Stakeholder definiert und sie dürfen nur mit Zustimmung des Stakeholders neu definiert, geändert oder erweitert werden.

Das Projekt wird mit den folgenden Artefakten abgegeben:

- Dokumentation:
  - Pflichtenheft: *To-Do-App_Pflichtenheft.docx*
- Software
  - Link zu GitHub Projekt: *https://github.com/sdozdogar/next-todo-app.git*
- Evidenz:
  - System/Software-Demo via Videoclip: *videoclip-link*

Anm.: Die Abgabetermine der Projetartefakts werden durch den Stakeholder festgelegt!

7. Projekt Meilensteine

Diese Kapitel beinhaltet die wichtigsten Meilensteine, die der Stakeholder festgelegt hat.

Folgende Meilensteine sind verbindlich definiert:
Meilenstein #1: Das Lastenheft ist fertiggestellt und mit dem Stakeholder abgestimmt.
Meilenstein #2: Das Pflichtenheft ist fertiggestellt und mit dem Stakeholder abgestimmt.
Meilenstein #3: (An diesem Meilenstein ist die Architektur im Vordergrund.)

- Komponenten-basierte Architektur ist entworfen und komponentenweise implementiert.
- Die externen Schnittstellen (Webservices) wurden entworfen/implementiert.
- Die GUI Komponente ist ansatzweise fertig.

Meilenstein #4: (An diesem Meilenstein ist das Testen im Vordergrund.)

- Die Test-Cases wurden aus den Use Cases abgeleitet und ansatzweise beschrieben.
- Die Testumgebung ist vorbereitet und ein Smoke Test ist durchgeführt.

Meilenstein #5

- Das Projekt ist per Vereinbarung abgegeben.

8. Referenzen

Diese Kapitel beinhaltet die wichtigsten Literaturquellen aufgelistet.