**arc42 Dokumentation**

**1. Einführung und Zielsetzung**

**1.1 Überblick über die Anforderungen**

SQS Project ist eine Anwendung zur Aufgabenverwaltung, die eine effiziente Aufgabenverwaltung über eine Webschnittstelle ermöglicht. Zu den wichtigsten Funktionen gehören das Hinzufügen von Aufgaben, das Rückgängigmachen erledigter Aufgaben, das Löschen von Aufgaben, das Markieren von Aufgaben als erledigt und die Gewährleistung der Sicherheit durch Eingabensanitisierung gegen XSS-Angriffe. Das Projekt legt auch Wert auf automatisierte Tests, um die funktionale Integrität zu gewährleisten.

**1.2 Qualitätsziele**

* **Benutzerfreundlichkeit:** Sorgen Sie für eine klare und intuitive Benutzeroberfläche für eine reibungslose Aufgabenverwaltung.
* **Verlässlichkeit:** Aufrechterhaltung einer konsistenten und korrekten Funktionalität bei allen Benutzerinteraktionen.
* **Leistung:** Optimieren Sie die Reaktionsfähigkeit der Anwendung, um mehrere Aufgaben effizient zu bewältigen.
* **Wartbarkeit:** Sicherstellen, dass die Codebasis sauber und einfach zu pflegen ist.
* **Sicherheit:** Die Anwendung sollte gegen gängige Schwachstellen, einschließlich XSS-Angriffen, geschützt sein, indem Benutzereingaben bereinigt und Abhängigkeiten regelmäßig überprüft werden.

**1.3 Stakeholders**

* **Entwickler:** Verantwortlich für die Implementierung und Wartung der Anwendung.
* **Benutzer:** Personen, die die Anwendung zur Verwaltung ihrer Aufgaben nutzen.

**2. Randbedingungen**

**2.1 Technische Randbedingungen**

* **Node.js-Version:** Die Anwendung ist kompatibel mit Node.js v20.12.2.
* **Datenbank:** Verwendet der Einfachheit halber eine In-Memory-SQLite3-Datenbank**.**
* **Frontend:** Verwendet EJS-Vorlagen für die Darstellung von Ansichten.
* **Sicherheit:** Implementiert die Eingabensanitisierung mit der xss-Bibliothek.

**2.2 Organisatiorische Randbedingungen**

**Zeitplan:** Das Projekt ist Teil einer Hausaufgabe mit bestimmten Fristen.

**Testen:** Erfordert automatisierte Tests mit Jest, Cypress, Artillery und Supertest.

**3. kontextabgrenzung**

**3.1 Fachlicher Context**

Das SQS-Projekt zielt darauf ab, die Produktivität zu steigern, indem es den Benutzern ein effektives Werkzeug zur Verwaltung ihrer täglichen Aufgaben an die Hand gibt. Es dient sowohl als pädagogische Übung in der Webentwicklung als auch als praktisches Hilfsmittel für die persönliche Aufgabenverwaltung.

**3.2 Technische kontext**

Bei der Anwendung handelt es sich um ein webbasiertes System, das auf Node.js und Express.js aufbaut, wobei die Datenspeicherung über SQLite3 erfolgt. Sie nutzt EJS für das serverseitige Rendering und integriert automatisierte Test-Frameworks für eine robuste Funktionsvalidierung.**3.3 Scope**

**In Scope:**

* Kernfunktionen für die Aufgabenverwaltung (hinzufügen, rückgängig machen, löschen, als erledigt markieren).
* Automatisierte Tests mit Jest, Cypress, Artillery und Supertest.
* Grundlegende Eingabesanitisierung gegen XSS-Angriffe.

**Out of Scope:**

* Advanced features such as user authentication.
* Deployment to a production environment.

**4. Lösungsstrategie**

**4.1 Entwicklung Ansatz**

* **Inkrementelle Entwicklung:** Entwickeln Sie Funktionen in kleinen, überschaubaren Schritten.
* **Modularer Code:** Modularer Aufbau der Codebasis für Übersichtlichkeit und Wartbarkeit.
* **Code-Qualität:** Verwenden Sie Prettier für eine konsistente Codeformatierung.

**4.2 Test Strategie**

* + **Unit Testing**: ich verwende Jest für Unit-Tests einzelner Funktionen und Module.
* Beispiel: Jest-Tests stellen sicher, dass die Kernfunktionen wie erwartet funktionieren.
  + **Integration Test**: ich Verwende Supertest, um API-Endpunkte und Interaktionen zwischen Komponenten zu validieren.
* Beispiel: Supertest-Tests überprüfen HTTP-Anfragen und -Antworten für CRUD-Operationen.
* **End-to-End-Tests**: Nutzen Sie Cypress für umfassende End-to-End-Tests von Benutzerinteraktionen.
* Beispiel: Cypress-Tests decken Szenarien wie das Hinzufügen und Löschen von Aufgaben über die Benutzeroberfläche ab.
* **Leistungstests**: Verwenden Sie Artillery, um die Anwendungsleistung unter Last zu bewerten.
* Beispiel: Artillery-Tests simulieren gleichzeitige Benutzerinteraktionen, um die Reaktionszeiten des Servers zu bewerten.

**4.3 Sicherheitsmaßnahmen**

* **Eingabesanitisierung:** Implementieren Sie Input-Sanitization mit der xss-Bibliothek, um XSS-Schwachstellen zu verhindern.
* **Verwaltung von Abhängigkeiten:** Aktualisieren Sie regelmäßig die Abhängigkeiten, um Sicherheitsrisiken zu minimieren.

**4.4 Strategie für den Einsatz**

* **Lokale Entwicklung:** Entwickeln und testen Sie lokal mit Node.js und SQLite3 für schnelle Iterationen.
* **Zukünftige Überlegungen:** Erwägen Sie die Bereitstellung in einer persistenten Datenbank für den Produktionseinsatz.

**5. Architektonische Entscheidungen**

**5.1 Entwurfsmuster**

Auf mein SQS-Projekt wurden keine spezifischen Entwurfsmuster angewandt.

**5.2 Technologies**

* **Node.js**: Ereignisgesteuerte JavaScript-Laufzeitumgebung für die serverseitige Entwicklung.
* **Express.js**: Schnelles Mikro-HTTP-Framework**.**
* **SQLite3:** Eingebettete SQL-Datenbank-Engine für die Datenspeicherung.
* **EJS:** Templating-Engine für das serverseitige Rendern von HTML-Ansichten.

**6. Übergreifende Konzepte**

Die folgenden übergreifenden Konzepte sind für das SQS-Projekt von Bedeutung:

1. **Gleichzeitigkeit:** Handhabung des gleichzeitigen Zugriffs auf Aufgaben und Daten**.**

**2. Bereinigung von Daten**: Sicherstellung der Datenintegrität durch Bereinigungsmechanismen.

**7. Qualitätsanforderungen**

**7.1 Leistung**

* **Reaktionsfähigkeit:** Stellen Sie sicher, dass Aufgaben zügig und ohne Verzögerungen erledigt werden.
* **Skalierbarkeit:** Effiziente Bewältigung einer wachsenden Zahl von Benutzern und Aufgaben.

**7.2 Sicherheit**

* **Schutz der Daten:** Schutz der Benutzerdaten vor unberechtigtem Zugriff und Manipulation.
* **XSS-Verhinderung:** Verhindern Sie XSS-Schwachstellen durch Bereinigung von Eingaben.

**7.3 Benutzerfreundlichkeit**

* **Intuitives Design:** Entwerfen Sie eine Schnittstelle, die einfach zu navigieren und zu verstehen ist.

**8. Risiken und technische Schulden**

Die Verwendung einer In-Memory-SQLite3-Datenbank ist zwar einfach und bequem, birgt aber das Risiko eines Datenverlusts, wenn der Server heruntergefahren wird. Um dieses Risiko zu mindern, ist es ratsam, sie durch eine persistente Datenbank wie PostgreSQL zu ersetzen.

**9. Glossar**

* **Artillerie**: Werkzeug für Leistungstests.
* **body-parser:** Middleware zum Parsen von HTTP-Anfragekörpern.
* **ejs:** Eingebettete JavaScript-Templating-Engine**.**
* **express:** Mikro-HTTP-Framework für Node.js.
* **sqlite3:** Asynchrone, nicht-blockierende SQLite3-Bindungen für Node.js.
* **xss:** Bibliothek zum Schutz vor XSS.
* **cypress:** End-to-End-Testing-Framework.
* **jest:** JavaScript-Test-Framework.
* **supertest:** HTTP-Assertions-Bibliothek.
* **prettier:** Code-Formatierer.