Projektarbeit BTS

Name der Projektarbeit: Ticket System

Abgabedatum: 16.12.2023

Schule: Christian-Schmidt-Schule

Lehrkraft: Heiko Zimmermann

Namen: Vinita Ganeshekumar, David-Ioannis Gugea

Eidesstaatliche Erklärung

Hiermit erklären wir, Vinita Ganeshekumar und David-Ioannis Gugea, dass wir die vorliegende Projektarbeit mit dem Thema „Ticket System“ selbstständig verfasst und nichts als die angegebenen Hilfsmittel benutzt haben. Die Stellen der Arbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angabe der Herkunft kenntlich gemacht. Dies gilt auch für Zeichnungen, Grafiken oder bildliche Darstellungen, sowie für Quellen des Internets. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen und wurde auch noch nicht veröffentlicht.

Ort Christian-Schmidt-Schule, Odenwaldstraße 5, 74172 Neckarsulm

Datum 16.12.2023

Unterschriften David-Ioannis Gugea, Vinita Ganeshekumar

Inhaltsverzeichnis

[1. Zielsetzung des Projektarbeits 4](#_Toc151641811)

[1.1 Pflichtenheft 4](#_Toc151641812)

[1.1.1 Zielbestimmungen 4](#_Toc151641813)

[1.1.2 Einsatz 4](#_Toc151641814)

[1.1.3 Produktübersicht 5](#_Toc151641815)

[1.1.4 Funktionen 5](#_Toc151641816)

[1.1.5 Qualitätsanforderungen 6](#_Toc151641817)

[1.1.6 Benutzeroberfläche 6](#_Toc151641818)

[1.1.7 Technisches Umfeld 6](#_Toc151641819)

[2. Projektplanung 6](#_Toc151641820)

[2.1 Gant-Diagramm, Meilensteine 6](#_Toc151641821)

[2.2 Datenbank ERM 6](#_Toc151641822)

[2.3 Programmiersprachen und Technologien 6](#_Toc151641823)

[3. Praktische Realisierung 7](#_Toc151641824)

[3.1 Containers 7](#_Toc151641825)

[3.2 Datenbank 7](#_Toc151641826)

[3.3 Backend 7](#_Toc151641827)

[3.4 Frontend 7](#_Toc151641828)

[4. Inbetriebnahme 7](#_Toc151641829)

[4.1 Voraussetzungen für die Inbetriebnahme 7](#_Toc151641830)

[4.2 Inbetriebnahme Schritte 7](#_Toc151641831)

[5. Ergebnis 7](#_Toc151641832)

[5.1 Fazit 7](#_Toc151641833)

[5.2 Ausblick 7](#_Toc151641834)

# Zielsetzung des Projektarbeits

## Pflichtenheft

### Zielbestimmungen

Die folgenden Features sind Teil der MUSS-Kriterien:

* Neue Konten eröffnen (Registrierung)
* Anmeldung mit Username und Passwort
* Tickets erstellen (mit Beschreibung)
* Tickets löschen
* Ticket Beschreibung ändern
* Tickets abschließen. Der User, die das Ticket abgeschlossen hat und das Datum, wann es abgeschlossen wurde, müssen ebenfalls abgespeichert werden.
* Tickets einer übersichtlichen Tabelle lesen können

Abgrenzungskriterien:

* Wenn ein User sich noch nicht angemeldet hat, darf er keinen Zugriff auf dem Ticket System haben. Man muss sich immer zuerst anmelden.
* Passwörter dürfen nicht im Klartext in der Datenbank gespeichert werden.

### Einsatz

Das Ticket-System kann in jedem Software-Entwicklungsprozess eingesetzt werden. Das gesamte Ticket-System kann containerisiert werden, so dass es in jeder Cloud-Umgebung laufen kann, was es sehr flexibel und einfach in der Anwendung macht. Die Datenbankvolumen können auch von Dritten verwaltet werden, was Backups erleichtert und das Datenbanksystem als Ganzes zuverlässiger macht. Das zugrundeliegende Datenbanksystem, PostgreSQL, kann auch mehrere Verbindungen gleichzeitig verarbeiten, so dass mehrere Nutzer gleichzeitig mit der Datenbank verbunden sein können, ohne sich um Race Conditions usw. kümmern zu müssen, da alles vom Server verwaltet wird.

### Produktübersicht

Man kann neuen Users erstellen und sich dann anmelden. Solange man nicht angemeldet ist, hat man keinen Zugriff auf das Ticket System

Tickets können mit einer Beschreibung erstellt werden. Sie können dann von bestimmten Mitgliedern der Gruppe abgeschlossen werden und ihre Beschreibung kann jederzeit geändert werden. Das Datum, an dem ein Ticket abgeschlossen wurde, wird auch gespeichert. Alles Ticket, unabhängig davon, wer sie erstellt hat, können in einer übersichtlichen Tabelle gelesen werden.

### Funktionen

In den folgenden Punkten, erfolgt die Erklärung jedes einzelnen Anwendungsfalls.

* User Registrierung: Wenn eine Person im System noch nicht angemeldet ist, muss ein neues Konto eröffnet werden. Das Frontend wird dann ein Request zu dem Backend senden, um zu überprüfen, ob die Benutzername und das Passwort gültig sind. Falls der eingetragene Benutzername und/oder Passwort sich bereits im Datenbanksystem befinden, wird eine Fehlermeldung angezeigt. Der User muss ein Konto mit einer neuen Benutzername erstellen.
* User Anmeldung: Der User trägt die Benutzername und das Passwort in den Inputfeldern ein und drückt auf „Login“. Das Backend wird dann überprüfen, ob die Benutzername und das Passwort gültig sind. Wenn sie gültig sind, bekommt der User Zugriff auf dem Ticket System. Wenn nicht, bekommt der User eine Fehlermeldung.
* Tickets lesen: Man kann als User alle Tickets in der Tabelle lesen.
* Ticket erstellen: Für die Erstellung eines Tickets, muss man eine Beschreibung einfügen. Das Frontend schickt dann zum Backend im Request Body die ID des eingeloggten Users und die Beschreibung des Tickets. Man wird dann immer wissen, wer das Ticket erstellt hat, wann, und worum es im Ticket geht. Das Datum, wann das Ticket erstellt wurde, wird im Backend automatisch gespeichert.
* Ticket Beschreibung ändern: Die neue Beschreibung des Tickets wird, zusammen mit dem Ticket ID, im Request Body abgespeichert und dann zum Backend abgeschickt. Das Backend wird dann die Beschreibung mit Hilfe eines ORMs. Die neue Beschreibung wird
* Ticket abschließen: Wenn ein Ticket abgeschlossen werden muss, wird das Backend die ticket id, und die user id von dem User, der das Ticket abschließen will, brauchen. Das Backend wird dann alle Daten in der Datenbank abspeichern, inklusiv, das Datum, wann ein Ticket abgeschlossen wurde. Alle anderen User werden dann sehen können wer das Ticket abgeschlossen hat, und wann es abgeschlossen wurde.
* Ticket löschen: Um ein Ticket komplett löschen zu können, wird das Backend die Ticket ID von dem Frontend benötigen. Man kann diese Funktion nicht rückgängig machen.

### Qualitätsanforderungen

|  |  |
| --- | --- |
| Änderbarkeit | Sehr gut |
| Effizienz | Sehr gut |
| Zuverlässigkeit | Sehr gut |
| Sicherheit | Gut |
| Benutzerfreundlichkeit | Sehr gut |
| Skalierbarkeit | Sehr gut |
| Wartbarkeit | Gut |
| Kompatibilität | Nicht relevant |
| Testbarkeit | Gut |
| Datenintegrität | Sehr gut |
| Performanz | Gut |

### Benutzeroberfläche

Die Benutzeroberfläche besteht aus zwei Hauptteilen: dem Anmelde- und Registrierungsformular und dem Ticketsystem. Die gesamte Anwendung ist mit einem Framework aufgebaut, das Single-Page-Anwendungen ermöglicht. Die Hauptvorteile von Single-Page-Anwendungen sind die schnelle Benutzererfahrung, die reibungslose Benutzerinteraktion, die reduzierte Serverlast, der verbesserte Arbeitsablauf für Entwickler und die stark verbesserte Benutzereinbindung.

### Technisches Umfeld

Die Softwaresysteme, die gebraucht werden, sind python und docker. Die docker compose Datei im root startet des gebrauchten Containers für das backend. Man kann dann mit venv eine virtuelle Umgebung starten und die benötigten python libraries installieren um den Backendserver starten zu können.

# Projektplanung

## Gant-Diagramm

A green line on a white grid

Description automatically generated

## Datenbank ERM

A graph with a line and a sign

Description automatically generated with medium confidence

## Programmiersprachen und Technologien

Wir haben uns für FastAPI als Backend-Framework entschieden, weil es über eine automatische API-Dokumentation verfügt, d. h. es generiert automatisch OpenAPI- und JSON-Schema-Dokumentation für die API auf der Grundlage der Python-Typ-Hinweise. Außerdem bietet es Typsicherheit, was bei der Erkennung von Fehlern und bei der Nutzung der API sehr hilfreich ist. Die einfache Integration mit pydantic macht es leicht, Daten zu validieren und Datenmodelle mit Validierungsregeln zu definieren und diese für kommende Anfragen zu verwenden.

SQLAlchemy ist das ORM (Object-Relational-Mapping), das wir als ORM gewählt haben, da es sich zu einem Industriestandard entwickelt hat und sehr einfach in FastAPI zu integrieren ist. Es macht die Interaktion mit relationalen Datenbanken sehr einfach und flexibel und ist besonders nützlich, wenn es um die Abstraktion von Datenbanken geht, da es mehrere Datenbank-Backends unterstützt und eine konsistente API unabhängig vom zugrunde liegenden Datenbanksystem bietet.

Wir haben uns vor allem deshalb für PostgreSQL entschieden, weil es Open Source ist und von der Community sehr gut unterstützt wird. Außerdem ist es sehr skalierbar und verfügt über eine Gleichzeitigkeitskontrolle für den Fall, dass mehrere Clients an die Datenbank angeschlossen sind. Wir haben die Postgresql-Datenbank und den pg-Admin-Browser in Docker-Containern ausgeführt. Ein Docker-Container ist ein leichtgewichtiges, portables Paket, das alles enthält, was zum Ausführen einer Software benötigt wird. Container sind wie virtuelle Maschinen, aber effizienter, da sie den Kernel des Host-Betriebssystems gemeinsam nutzen und gleichzeitig eine Isolierung für die Anwendung bieten. Wir haben dies aus Gründen der Isolation und Portabilität getan. Durch die Verwendung der Docker-Engine ist es einfach und flexibel, sie überall zu starten, was die Entwicklung einfacher und schneller macht.

# Praktische Realisierung

## Containers

Ein Container ist ein Linux-Prozess, der durch cgroups und Namespaces verwaltet wird. Docker verwendet ein Linux-Subsystem auf Windows, um die Verwendung von Containern auf Windows-Systemen zu ermöglichen.

Ein Prozess ist die Instanz eines Programms, das gerade ausgeführt wird. Jeder Prozess enthält seinen eigenen Heap (Speicher). Ein Prozess enthält auch Prozessregister, Programmzähler, Stapelzeiger und, wie bereits gesagt, "Speicherseiten" (d.h. Stapelhaufen, statischer Speicher usw.). Da jeder Prozess seinen eigenen Speicheradressraum (Heap) hat, bedeutet dies auch, dass ein Prozess andere Prozesse nicht beschädigen kann, d.h. wenn ein Prozess ausfällt, beschädigt dies nicht andere laufende Prozesse. Ein Prozess kann mehrere andere Prozesse starten, wobei der erstellende Prozess als Elternprozess bezeichnet wird und die von diesem Prozess gestarteten Prozesse als seine Kindprozesse. Jeder erstellte Prozess kann dann wiederum neue andere Prozesse erstellen, die dann einen Baum von Prozessen bilden. Jedes Mal, wenn Sie einen Linux-Rechner starten, wird ein erster Prozess namens "INIT" gestartet, der die PID (Prozess-ID) 1 erhält. Hier beginnt der Prozessbaum.

Mit Cgroups können Prozesse in Gruppen organisiert werden. Diese Gruppen können dann überwacht werden und es können bestimmte Grenzen für diese Gruppen festgelegt werden (z. B. CPU-/Speichergrenzen). Cgroups werden nicht automatisch erstellt. Eine Cgroup kann für bestimmte Prozesse und den Prozessbaum, der aus ihnen hervorgeht, erstellt werden.

Namensräume werden verwendet, um Prozesse vom Rest zu isolieren. Sie können verwendet werden, um Ressourcen wie Prozess-IDs, Netzwerkschnittstellen usw. zu partitionieren, so dass verschiedene Prozesse ihre eigenen Instanzen dieser Ressourcen haben können, als ob sie in ihrer eigenen unabhängigen Umgebung laufen würden.

Wir haben uns aus den folgenden Gründen für die Verwendung von Containern in unserem Projekt entschieden:

* Isolierung: Anwendungen können unabhängig voneinander arbeiten, ohne sich gegenseitig zu stören, da das Dateisystem und die Prozesse in Containern voneinander getrennt sind. Dadurch, dass sichergestellt wird, dass Konfigurationen und Abhängigkeiten innerhalb des Containers enthalten sind, trägt diese Isolierung zur Minimierung von Konflikten und Kompatibilitätsproblemen bei.
* Portabilität: Die Anwendung und ihre Abhängigkeiten sind in Containern gekapselt, was den Transport und den Betrieb der Anwendung in verschiedenen Kontexten, einschließlich Entwicklung, Test und Produktion, erleichtert. Das Problem "es funktioniert auf meinem Rechner" wird durch diese Portabilität gelöst, die auch das Bereitstellungsverfahren vereinfacht.
* Schnelles Testen und Feedback: Mit Containern können Entwickler Änderungen lokal in einer produktionsähnlichen Umgebung testen. So können Ideen schnell getestet werden und man erhält sofort Feedback, was einen iterativen Entwicklungsprozess fördert.

## Datenbank

In einer relationalen Datenbank können Daten aufgrund der Beziehungen zwischen den Daten dank ihres strukturellen Aufbaus angeordnet und zugänglich gemacht werden.

In relationalen Datenbanken werden die Daten in Tabellen gespeichert, die in Zeilen und Spalten angeordnet sind. In einer Tabelle steht jede Zeile für einen Datensatz und jede Spalte für ein Feld oder Attribut des Datensatzes. Gemeinsame Felder (oder Schlüssel) in vielen Tabellen dienen als Grundlage für Beziehungen zwischen ihnen. Dank dieser Verknüpfungen kann die Datenbank verwandte Daten aus anderen Tabellen effektiv miteinander verbinden. Eine Spalte (oder Gruppe von Feldern) in einer Tabelle, die eine eindeutige Identität für jeden Datensatz in der Tabelle bietet, wird als Schlüssel bezeichnet. Es gibt verschiedene Arten von Schlüsseln, z. B. Fremdschlüssel, die zur Erstellung tabellenübergreifender Verknüpfungen verwendet werden, und Primärschlüssel, die zur eindeutigen Identifizierung von Datensätzen innerhalb einer Tabelle dienen.

Die Verwendung einer relationalen Datenbank bietet verschiedene Vorteile, darunter:

* Datenintegrität: Einschränkungen wie Primärschlüssel, Fremdschlüssel und eindeutige Einschränkungen werden von relationalen Datenbanken verwendet, um die Datenintegrität zu gewährleisten. Dies trägt dazu bei, die Daten konsistent und genau zu halten.
* Beziehungen: Relationale Datenbanken erleichtern die Erstellung von Verbindungen zwischen Tabellen. Dies ermöglicht es, komplizierte Datenstrukturen auszudrücken und zugehörige Daten effektiv abzufragen.
* Skalierbarkeit: Relationale Datenbanken können horizontal (durch Verteilen der Datenbank auf mehrere Server) oder vertikal (durch Hinzufügen weiterer Ressourcen zu einem einzigen Server) erweitert werden. Daher können sie steigende Benutzerzahlen und Datenmengen bewältigen.
* ACID-Eigenschaften: Relationale Datenbanken folgen den ACID-Eigenschaften (Atomicity, Consistency, Isolation, Durability), die eine konsistente und verlässliche Transaktionsverarbeitung auch bei Systemausfällen garantieren.

Postgresql ist eine relationale Datenbank, die skalierbar ist, Datenintegrität bietet und den ACID-Standards entspricht, weshalb wir sie für unsere Datenbank ausgewählt haben.

Die Datenbank besteht aus zwei Tabellen: Benutzer und Tickets. Die Tabelle "Benutzer" enthält einen Primärschlüssel, nämlich die ID des Benutzers. Sie enthält außerdem den Benutzernamen, das Passwort, den Benutzertyp und den Salt-Wert, der zum Hashing des Passworts verwendet wird. Beim Hashing werden die Eingabedaten, die häufig eine variable Größe haben, in eine Zeichenkette fester Länge umgewandelt, die in der Regel als zufällige Folge von Buchstaben und Ziffern erscheint. Eine Methode zur Erhöhung der Sicherheit von gehashten Kennwörtern ist das Salting. Eine zufällige Datenfolge, die speziell für jeden Benutzer erstellt wird, wird als Salt bezeichnet. Das Kennwort des Benutzers und das Salt werden dann vor dem Hashing gemischt. Das Salt und der generierte Hash werden in der Datenbank gespeichert.

Die Ticket-Tabelle enthält die ID des Tickets, die ein Primärschlüssel ist. Sie enthält die Beschreibung, das Erstellungs- und Abschlussdatum sowie einen booleschen Wert namens "done", der anzeigt, ob das Ticket abgeschlossen wurde oder nicht. Sie enthält auch zwei Primärschlüssel zur Benutzertabelle, "closed\_by", der auf die ID des Benutzers verweist, der das Ticket geschlossen hat, und "user\_id", der auf den Benutzer verweist, der das Ticket erstellt hat.

## Backend

Mit Hilfe des Python Fast-Api-Frameworks bietet das Backend eine Restful-API. Die Representational State Transfer Application Programming Interface (REST API) ist ein architektonischer Ansatz für die Entwicklung von Netzwerkanwendungen. Es handelt sich um eine Sammlung von Richtlinien und Beschränkungen, die bei der Entwicklung von Online-Diensten verwendet werden. REST ist eine Reihe von Prinzipien, die angeben, wie Webdienste erstellt werden sollten; es ist weder ein Standard noch ein Protokoll an sich.

Zu den wichtigsten Grundsätzen und Merkmalen einer REST-API gehören:

* Zustandslosigkeit: Jede Anfrage, die ein Client an einen Server sendet, muss alle Details enthalten, die der Server benötigt, um die Anfrage zu verstehen und zu beantworten. Zwischen den Anfragen sollte der Server den Status des Clients nicht verfolgen.
* Client-Server-Architektur: Durch die Verwendung eines Netzwerks zur Erleichterung der Kommunikation sind der Client und der Server unabhängige Einheiten. Während der Server für die Bearbeitung von Anfragen und die Ressourcenverwaltung zuständig ist, ist der Client für die Benutzeroberfläche und die Benutzererfahrung verantwortlich.
* Einheitliche Schnittstelle: Eine konsistente und einheitliche Schnittstelle ist ein Muss für REST-APIs. Dies umfasst die Verwendung gängiger HTTP-Methoden (GET, POST, PUT, DELETE) sowie eine Reihe von Regeln für die Darstellung und Benennung von Ressourcen.

Ein Web-Framework ist eine Softwarearchitektur, die die Entwicklung von Webanwendungen unterstützen soll. Dazu bietet es vorgefertigte Teile und Dienstprogramme zur Vereinfachung von Routinevorgängen wie der Beantwortung von HTTP-Anfragen, der Pflege von Datenbanken und der Vereinfachung der Gestaltung von Benutzeroberflächen. Web-Frameworks bieten eine definierte Basis und wiederverwendeten Code, damit Entwickler Webanwendungen schneller erstellen können.

Die Hauptgründe, warum wir uns für fast api entschieden haben, sind die automatische Dokumentation, die Handhabung von Typ-Annotationen und die Erleichterung der Datenvalidierung und Serialisierung mit Pydantic, was den Entwicklungsprozess rationalisiert. Außerdem ermöglicht sie Dependency Injection, was die saubere, modulare Organisation und Verwaltung von Abhängigkeiten vereinfacht.

Wir haben auch ein ORM verwendet und es in das Web-Framework integriert, um mit der Postgresql-Datenbank zu arbeiten. ORM steht für Object-Relational Mapping. Es handelt sich um ein objektorientiertes Programmierparadigma und eine Programmiertechnik, die die Lücke zwischen relationalen Datenbanken und Software-Entwurfsmustern in der Softwareentwicklung schließt.

Mit ORM-Frameworks können Sie zwischen der relationalen Darstellung in der Datenbank und der objektorientierten Darstellung im Code konvertieren und Datenbanktabellen auf Klassen abbilden. Wir haben uns für SQLAlchemy entschieden, weil es datenbankunabhängig und flexibel ist und sich sehr einfach in eine schnelle API integrieren lässt.

## 3.4 Frontend

# Inbetriebnahme

## Voraussetzungen für die Inbetriebnahme

Die Software, die zum Starten des Backends benötigt wird, ist Docker Desktop und Python.

## .Inbetriebnahme Schritte

Um das Backend zu starten, müssen der Docker-Desktop und Python installiert sein. Anschließend muss im Stammverzeichnis der Anwendung der Befehl "docker-compose up -d" ausgeführt werden, um die für die Datenbank benötigten Container zu starten. Nach dem Start des Containers muss das venv in dem Ordner aktiviert werden, in dem das Backend gespeichert wurde. Nach der Aktivierung von venv müssen alle Pakete aus der Datei requirements.txt installiert werden. Nach der Installation kann der Backend-Api-Server mit dem Befehl: uvicorn main:app gestartet werden.

# Ergebnis

## Fazit

## Ausblick