Luca Köppel & Simon Hefti

Distributed Systems

FS 23 – Challenge Task

1. Inhalt

2. Beschreibung

Unser Projekt "Bites & Sips" ist eine Web-Applikation, die es Benutzern ermöglicht, Restaurants und Bars in ihrer Nähe anzuzeigen. Die Anwendung verwendet die Google API, um Standorte basierend auf den benutzerdefinierten Filtern zu suchen. Die Suchergebnisse werden mithilfe eines Algorithmus bewertet und nach Kriterien wie Preis und Distanz sortiert. Die Ergebnisse werden dem Benutzer in Form von Karten präsentiert, zwischen denen er navigieren kann. Benutzer können Vorschlägen positive oder negative Bewertungen geben, um die Anzeigereihenfolge zu beeinflussen.

3. Architektur

Unsere Web-Applikation besteht aus zwei Flask Service Instanzen als Backend und einer Vue Instanz als Frontend. Der Caddy-Loadbalancer nimmt alle Anfragen entgegen und verteilt den Datenverkehr auf die Flask-Instanzen sowie das Vue-Frontend. Dabei werden alle Anfragen auf die Route /api auf die Flask-Instanzen (Ports 5000/5001) verteilt, um Skalierbarkeit und Zuverlässigkeit sicherzustellen. Alle anderen Routen werden zum Frontend weitergeleitet (Port 8080).

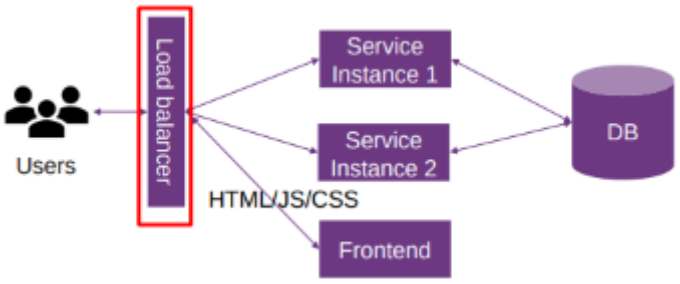
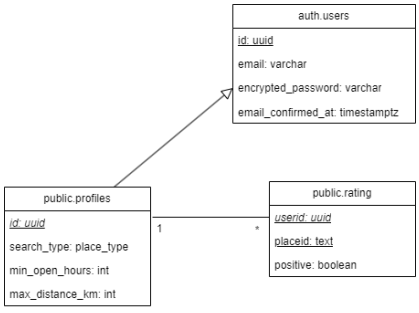


Abbildung 1: Architektur

Die Daten werden in einer Supabase-Datenbank gespeichert.

4. Technologien

- **Backend**: Das Backend basiert auf Python Flask, einem Web-Framework, das die Verarbeitung von Benutzeranfragen auf bestimmte Routes ermöglicht. Diese Routes sind im File BytesAndSips.py definiert. Es verwendet die Bibliotheken und Numpy für die Datenmanipulation und mathematische Operationen. Für die API, die Datenbank und das Generieren der JWT werden verschiedene Keys benötigt. Diese werden zu Beginn mit load_dotenv() in die Umgebungsvariablen geladen. Dafür muss ein .env-File im Backend-Ordner vorhanden sein.
- **Frontend**: Das Frontend wird mit Vue umgesetzt, einem JavaScript-Framework zur Erstellung interaktiver Benutzeroberflächen.
- **Datenbank**: Wir nutzen die Supabase-Datenbank, die auf PostgreSQL basiert, zur effizienten Speicherung und Verwaltung der Daten. Supabase bietet reibungslose Interaktion mit unserer Anwendung, ohne dass wir uns um die komplexen Aspekte der Datenbankverwaltung kümmern müssen, zudem bietet die Python API eine zuverlässige Schnittstelle für Python inklusive Benutzerverwaltung mit E-Mail-Registrierung. Die Datenbank-Verbindung wird im File db_requests.py hergestellt. In der Datenbank werden die Benutzer, ihre letzten Sucheinstellungen sowie ihre Bewertungen gespeichert.



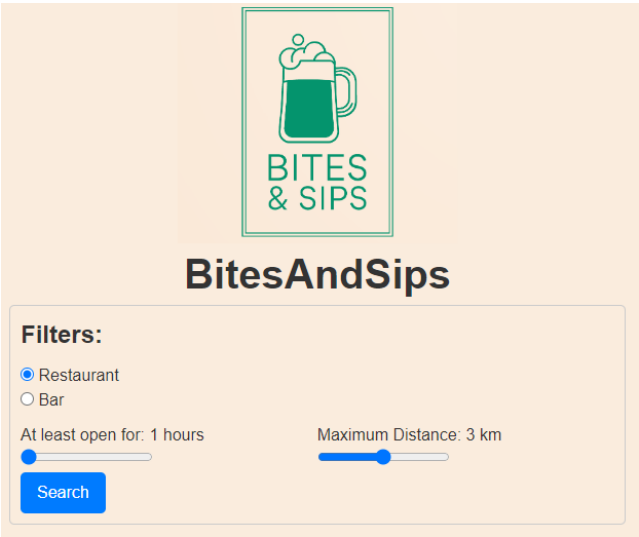
- **Load-Balancer**: Der Caddy-Loadbalancer verteilt den Datenverkehr auf die Flask-Instanzen, um die Last gleichmäßig zu verteilen und die Skalierbarkeit zu verbessern. Caddy ist sehr praktisch, da wenig Konfiguration nötig ist und http/3 sowie HTTPS in den neusten Versionen automatisch aktiviert sind. Der Load-Balancer ist so konfiguriert, dass eine Service-Instanz nach einem fehlerhaften Aufruf für 30 Sekunden nicht mehr verwendet wird.

Abbildung 2: DB-Klassendiagramm

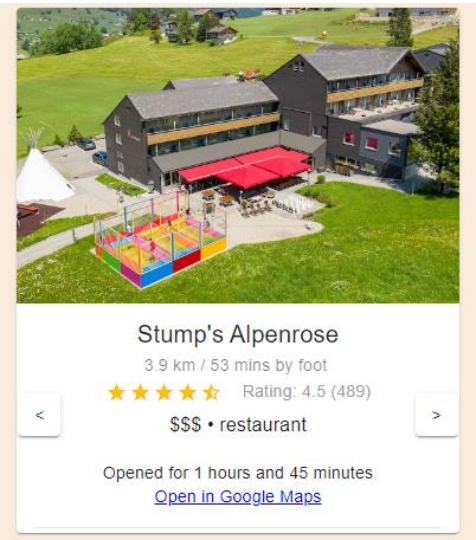
- **Container & Deployment**: Docker wird verwendet, um die Anwendung und ihre Abhängigkeiten in isolierten Containern zu verpacken und eine einfache Bereitstellung zu ermöglichen. Das docker-compose-File befindet sich im Ordner Resources, um die Container zu starten muss deshalb auf den Unterordner Resources navigiert werden. Dann kann man den Container mit dem «docker compose up --build» Befehl starten.
- **Authentifizierung**: Die Authentifizierung erfolgt mit JSON Web Tokens (JWT), um eine sichere Kommunikation zwischen Benutzer und Backend zu gewährleisten. Die Benutzer werden in der Datenbank gespeichert. Über das Web-UI können sich Benutzer mit ihrer E-Mail-Adresse registrieren. Sie erhalten per E-Mail einen Link, über welchen sie den Account aktivieren können. Dabei wird manchmal aufgrund der E-Mail-Filter gewisser Anbieter ein Fehler angezeigt, die Aktivierung funktioniert aber trotzdem. Eine Sitzung ist eine Stunde lang gültig.
- **API**: Die Google Maps API (Places, Geocoding, Distance) wird verwendet, um Informationen über nahegelegene Orte, Öffnungszeiten und Gehstrecken abzurufen. Alle Google-API-Anfragen werden im File place_requests.py durchgeführt. Die Daten werden schliesslich mithilfe der Python-Libraries Pandas und Numpy gefiltert und erweitert.

5. Ergebnis

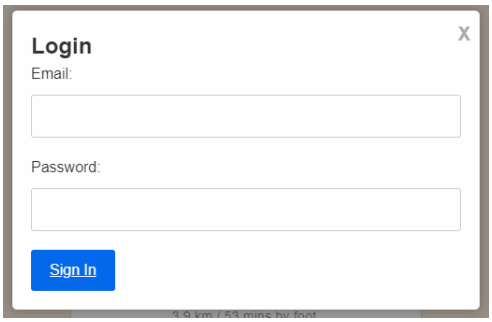
Filter



Ergebnis



Login



6. Reflexion

Positiv:

- Erfahrung mit der Entwicklung des Python-Backends und Vue-Frontends
- Verwendung des benutzerfreundlichen Flask-Frameworks
- Nutzung von Supabase für effiziente Datenverwaltung

Schwierigkeiten:

- Herausforderungen bei der Caddy-Einrichtung, insbesondere mit HTTP3
- Anfangsschwierigkeiten bei der Verwendung von Docker
- Komplexität der Authentication-Implementierung

Lernfortschritte:

- Erwerb von Full-Stack-Entwicklungsfähigkeiten
- Erfahrung im Umgang mit Docker zur Containerisierung der Anwendung

Verbesserungen:

- Weniger Google Places API abfragen pro Request
    - Da wir momentan pro Empfehlung-Set 5-20 Abfragen brauchen…