# Projektplanung: Kleinanzeigen-Webseite „GenuineGoods“

## 1. Projektkonzept

### 1.1 Konzept der Webanwendung und Zielgruppe?

Die Webanwendung dient als Plattform für Kleinanzeigen, auf der Nutzer Artikel oder Dienstleistungen anbieten, suchen und miteinander in Kontakt treten können.

Die Hauptzielgruppe sind Privatpersonen und kleine Unternehmen, die Artikel oder Dienstleistungen anbieten oder suchen möchten.

### 1.2 Welche Probleme löst die Anwendung?

* **Marktplatz für Angebote**: Sie bietet einen zentralen Ort, an dem Nutzer ihre Artikel oder Dienstleistungen auflisten können.
* **Einfache Kommunikation**: Mit integrierter Chat-Funktion können Käufer und Verkäufer direkt miteinander kommunizieren.
* **Zielgerichtete Suche**: Durch Such- und Filteroptionen können Nutzer gezielt nach bestimmten Artikeln oder Dienstleistungen suchen.

## 1.3 Welche Technologien wurden verwendet und warum?

### 1.3.1 Backend: FastAPI

* **Performance**: Durch die Verwendung von Starlette und Pydantic eines der schnellsten Frameworks für das Erstellen von APIs mit Python
* **Einfache Validierung**: Automatisierung der Eingabevalisierung durch die Verwendung von Pydantic
* **Eingebaute OAuth- und JWT-Authentifizierung**: Erleichtert die Implementierung sicherer APIs
* **Automatische API-Dokumentation**: Bietet automatische interaktive API-Dokumentation über Swagger UI
* **Asynchroner Support**: Unterstützt sowohl asynchrone als auch synchrone Anfragen

### 1.3.2 Datenbank: PostgreSQL

* **Erweiterbarkeit**: Unterstützt sowohl JSON als auch relationale Datenmodelle
* **Stabilität und Zuverlässigkeit**: Hohe Zuverlässigkeit und Datenintegrität
* **Konformität zu SQL-Standards**: Hält SQL-Standards ein
* **Open Source**

### 1.3.3 Frontend: React JS

* **Intuitiv**: Einfache und intuitive Erstellung von Code
* **Komponentenbasierte Architektur**: Ermöglicht Wiederverwendbarkeit und Testbarkeit von Code, Verkürzung der Entwicklungszeit
* **Virtual DOM**: Verwendet Virtual DOM, wodurch nur der notwendige Teil des Programms und nicht das Ganze aktualisiert wird, dadurch schnellere Entwicklung möglich
* **Flexibilität**: Modulare Struktur und große Anzahl von Bibliotheken erleichtern die individuelle Anpassung der Entwicklung und deren Wartung und Skalierbarkeit

### 1.4 Welche Funktionen bietet die Anwendung?

* **Angebote erstellen und verwalten**: Nutzer können Artikel oder Dienstleistungen auflisten, bearbeiten oder löschen.
* **Kategorien & Unterkategorien**: Artikel oder Dienstleistungen können bestimmten Kategorien und Unterkategorien zugeordnet werden, um die Navigation und Suche zu erleichtern.
* **Suchfunktion**: Ermöglicht das Durchsuchen der Angebote basierend auf verschiedenen Kriterien.
* **Chat-Funktion**: Ermöglicht den direkten Kontakt zwischen Käufern und Verkäufern.
* **Folgen**: Nutzer können Favouriten markieren.

## 2. Architektur der Anwendung

### 2.1 Struktur und Komponenten

**Frontend**: Die Benutzeroberfläche, über die Nutzer mit der Anwendung interagieren.

**API**:

* **Endpoints**: Enthält die verschiedenen API-Endpunkte, über die die Frontend-Anfragen bearbeitet werden.
* **Crud**: Enthält die CRUD-Operationen (Create, Read, Update, Delete) für die verschiedenen Datenbanktabellen.
* **Schemas**: Definiert die Pydantic Schemas für die Kommunikation über die API.
* **Search**: Beinhaltet die Funktionalität, um die Angebote basierend auf verschiedenen Kriterien zu durchsuchen.
* **Websockets**: Ermöglicht die Kommunikation für den Chat durch die Klasse *ConnectionManager*.
* **Utils**: Bietet Hilfsfunktionen und -skripte (FileOperations, Konfigurationen).

**DB\_Queries**: Enthält die spezifischen Datenbankabfragen und -operationen, die von den CRUD-Operationen und anderen Teilen der Anwendung verwendet werden.

**Datenbank**: In der PostgreSQL Datenbank *web2\_db* werden in den 6 Tabellen *categories*, *chats*, *followings*, *offers*, *subcategories* und *users* alle Daten gespeichert.

### 2.2 Interaktion der Komponenten

Das **Frontend** sendet Anfragen an die **API-Endpoints** im **Backend**, um Daten abzurufen oder Aktionen auszulösen (z. B. einen Artikel suchen, einen Chat starten).

Die **API** verarbeitet diese Anfragen über die **Endpoints**, führt über die **CRUD-Operationen** die entsprechenden Datenbankabfragen durch und gibt die Ergebnisse an das **Frontend** zurück.

Die **CRUD-Operationen** interagieren mit der **Datenbank** über die SQLAlchemy ORM Models (**models.py**).

Der **Websockets-Endpunkt** für den Chat ermöglicht eine bidirektionale Kommunikation zwischen Nutzern.