Booking App

DJANGO – TAILWIND - POSTGRES

SILAS GABRIEL ENZLER

Inhalt

Technologie Stack

Name	Version	Verwendung
Python	3.12	Programmiersprache
Django	6.0	Webframework
Postgres	18	Datenbank
Tailwind	4.0	CSS UI-styling
Docker	28.1.1	Containerisierung
Docker compose	-	Container generierung

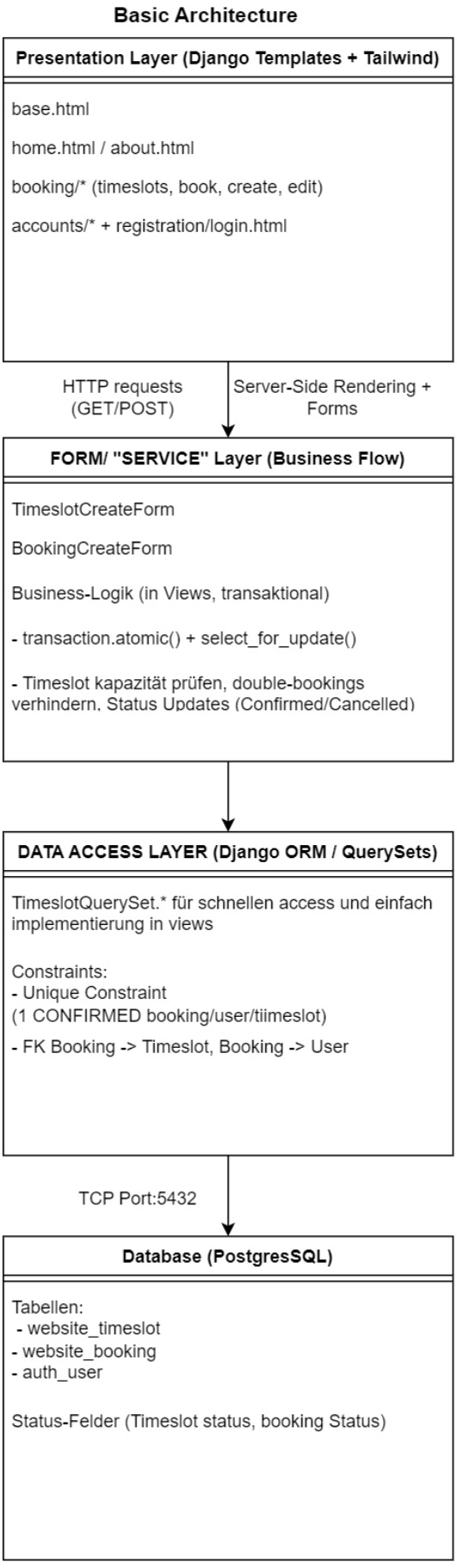
Die Anwendung wurde mit Python 3.12 als Programmiersprache umgesetzt und nutzt Django 6.0 als zentrales Webframework für Backend-Logik, Routing und serverseitiges Rendering.
Die Persistierung der Daten erfolgt in einer PostgreSQL-Datenbank (Version 18), die über das Django ORM angebunden ist.
Für das UI-Styling wird Tailwind CSS 4.0 eingesetzt, wodurch eine konsistente und moderne Benutzeroberfläche entsteht.
Die gesamte Anwendung ist containerisiert und wird mithilfe von Docker und Docker Compose betrieben, was eine reproduzierbare und isolierte Laufzeitumgebung ermöglicht.

Die Anwendung nutzt Django als zentrales Webframework und setzt das Model-View-Template-Pattern um.
Models (models.py) definieren die Datenstrukturen und Geschäftsregeln, während Views (views.py) die HTTP-Anfragen verarbeiten und die Geschäftslogik steuern.
Formulare (forms.py) übernehmen Eingabevalidierung und Datentransformation, bevor Änderungen persistiert werden.
Authentifizierung, Sitzungsverwaltung und Sicherheit werden vollständig durch die integrierten Django-Mechanismen (Django.auth) umgesetzt.

Django ist als modulares Webframework aufgebaut, bei dem die grundlegende Projektkonfiguration zentral in der Datei config/settings.py definiert wird.
Dort werden unter anderem installierte Applikationen, Middleware, Datenbankverbindungen und Sicherheitsoptionen konfiguriert.
Das Projektgerüst sowie Verwaltungsbefehle wie Datenbankmigrationen oder das Starten des Entwicklungsservers werden über das automatisch generierte Skript manage.py gesteuert.
Die URL-Verarbeitung beginnt in der zentralen config/urls.py, von wo aus die Routen logisch auf die jeweiligen App-spezifischen urls.py Dateien verteilt werden.
Diese Struktur ermöglicht eine klare Trennung von Verantwortlichkeiten und eine skalierbare Organisation der Anwendung nach funktionalen Modulen.

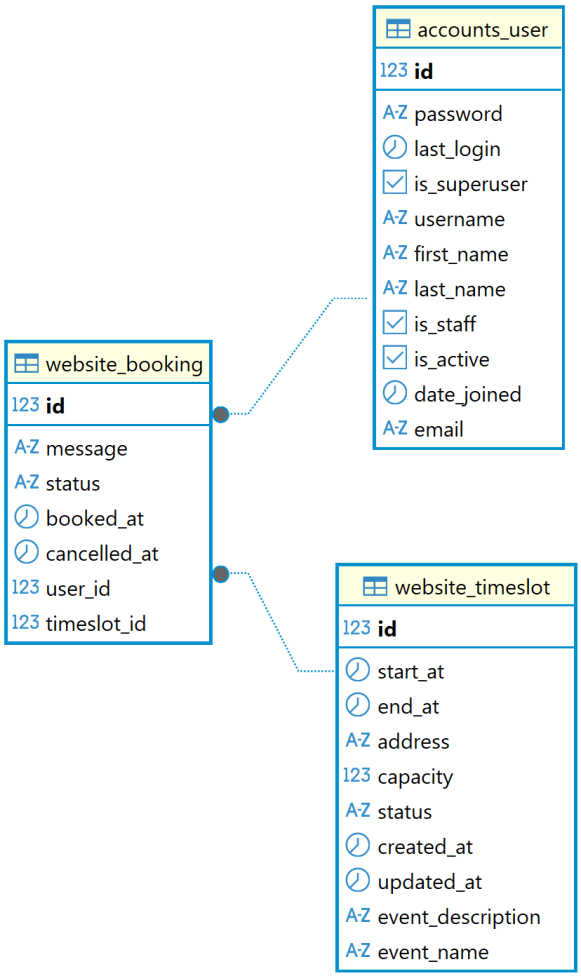
Backend Architektur

Die Anwendung folgt einer klaren, schichtbasierten Architektur mit serverseitigem Rendering.
Die Präsentationsschicht besteht aus Django Templates und verarbeitet Benutzerinteraktionen über klassische HTTP-GET- und POST-Requests.
Die Geschäftslogik wird in Views und Formularen umgesetzt und nutzt transaktionale Mechanismen, um Konsistenz und Datenintegrität sicherzustellen.
Der Datenzugriff erfolgt über das Django ORM auf eine PostgreSQL-Datenbank, die in einem separaten Container über das native PostgreSQL-Protokoll angebunden ist.



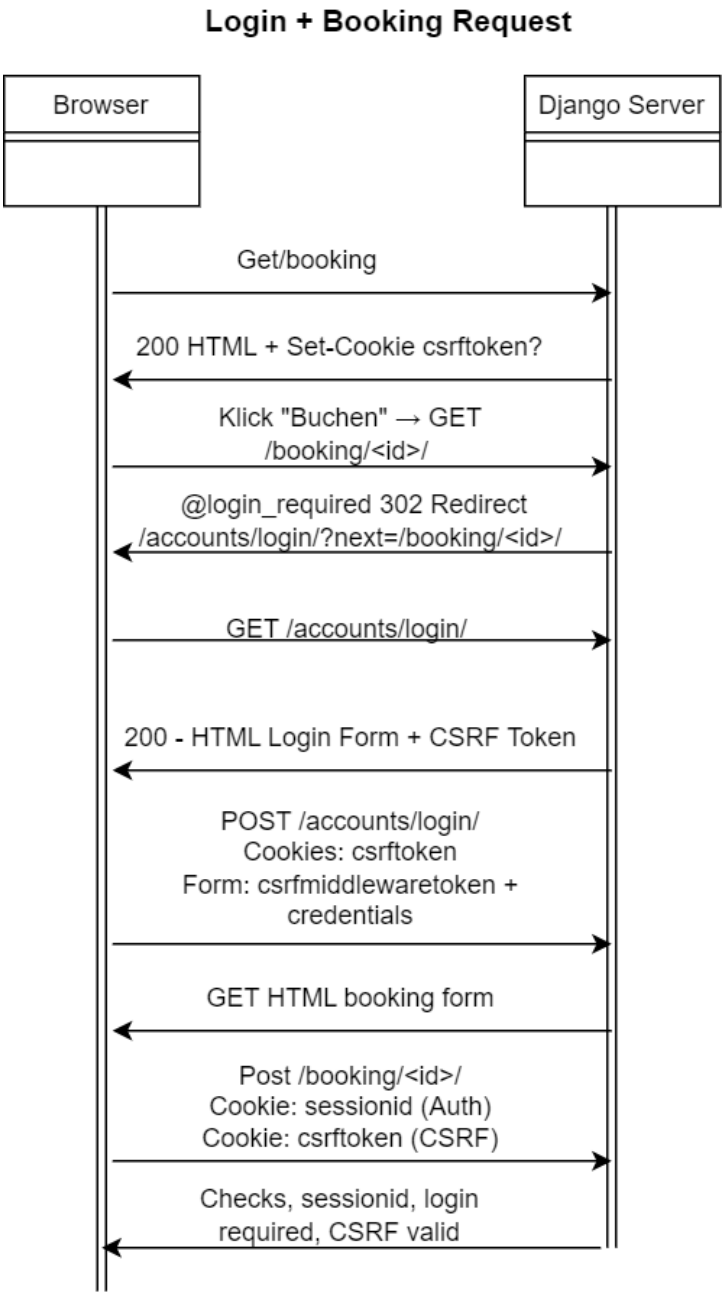
Datenbank ERD

Das Entity-Relationship-Diagramm zeigt die zentralen Datenstrukturen der Anwendung und deren Beziehungen.
Ein Benutzer (accounts_user) kann mehrere Buchungen (website_booking) besitzen, während jede Buchung eindeutig einem Benutzer und einem Timeslot (website_timeslot) zugeordnet ist.
Die Tabelle website_timeslot speichert zeitliche und organisatorische Informationen zu Events, einschließlich Kapazität und Status.
Status- und Zeitstempelfelder ermöglichen die Nachverfolgung von Buchungen sowie die Unterscheidung zwischen aktiven und abgesagten Events.



Authentifizierungs Flow

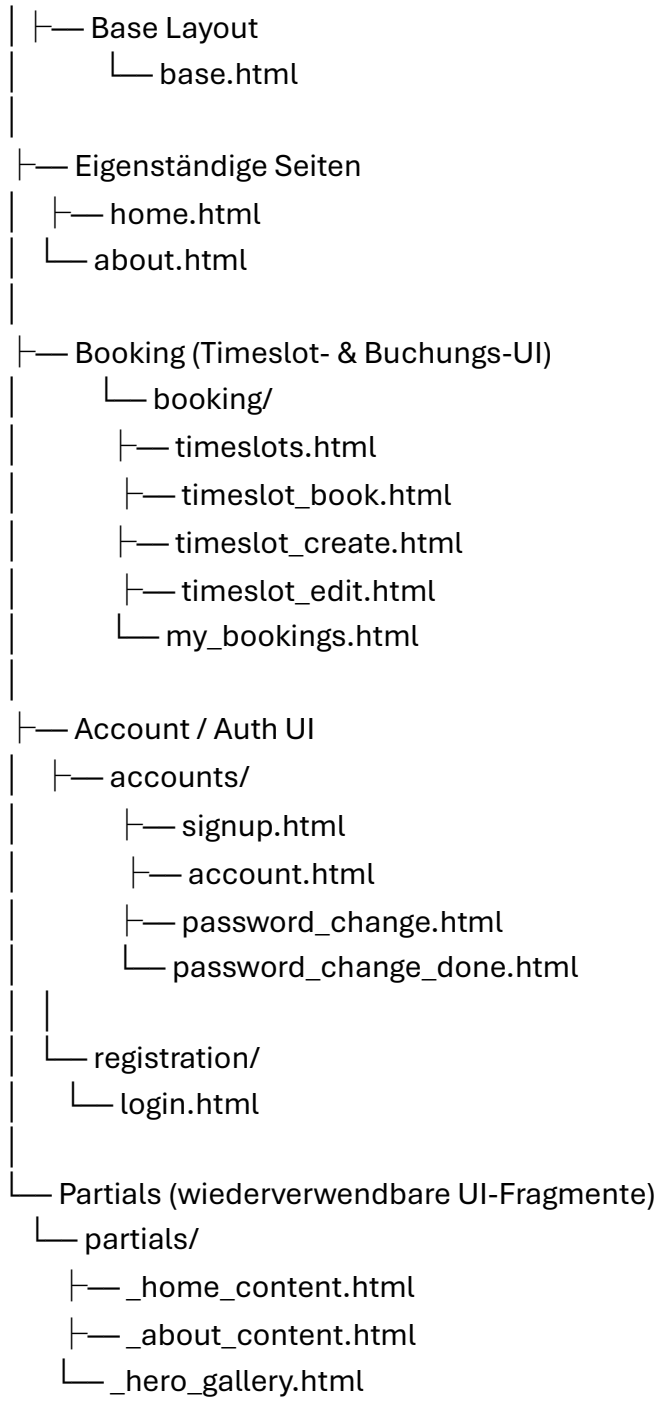
Beim Zugriff auf den Buchungsprozess wird jeder Request zunächst durch die Django SessionMiddleware und AuthenticationMiddleware verarbeitet, welche anhand des sessionid-Cookies den aktuell eingeloggten Benutzer als request.user identifizieren.
Der Zugriff auf die Buchungs-Views ist durch login_required abgesichert, sodass nicht authentifizierte Benutzer automatisch zur Login-Seite weitergeleitet werden.
Zustandsverändernde Buchungsanfragen werden zusätzlich durch die CsrfViewMiddleware geschützt, indem das im Formular übermittelte CSRF-Token mit dem Cookie-Token verglichen wird.
Erst wenn Session-Authentifizierung, Benutzerzuordnung und CSRF-Prüfung erfolgreich sind, wird die Buchungslogik ausgeführt und die Aktion eindeutig dem erwarteten Benutzer zugeordnet.



State-Management Flow

In dieser Anwendung wird der Zustand vollständig serverseitig durch Django verwaltet.
Der Authentifizierungszustand wird über Sessions gespeichert und bei jedem Request durch die Django Middleware als request.user bereitgestellt, während fachliche Zustände wie Buchungen dauerhaft in der Datenbank persistiert sind.
Views lesen diesen Zustand bei jedem Request neu aus und erzeugen daraus den aktuellen UI-Zustand, der serverseitig gerendert wird.
Im Gegensatz dazu verwalten React-Anwendungen ihren Zustand primär im Browser über globale States oder Contexts, während Django bewusst auf serverseitiges State Management setzt.

Frontend Architektur (Django Templates – Server-Side Rendering)



Die Benutzeroberfläche der Anwendung ist serverseitig gerendert und basiert auf Django Templates. Es wird kein Frontend Framework benützt.

Ein zentrales Layout-Template (base.html) definiert die grundlegende Seitenstruktur, während spezifische Seiten-Templates nach fachlichen Domänen wie Booking und

Accounts organisiert sind.

Wiederverwendbare UI-Bestandteile sind als Partials ausgelagert und werden bei Bedarf in Seiten eingebunden, unter anderem zur Unterstützung von HTMX (welches ich noch implementieren werde).
Es existiert kein separates Frontend mit eigenem Routing, die Navigation und Interaktion erfolgen vollständig über das Django-Backend.

Die Booking html's könnten auch noch besser aufgeteilt und in partials aufgeteilt werden. Sodass am ende HTMX in den meisten Fällen genutzt wird, um partials einzusetzen und die Seite fast nie vollständig neu geladen werden muss.

API Integration

In React-Anwendungen kommunizieren UI-Komponenten über dedizierte Service- und API-Client-Schichten mit einem separaten Backend, wobei HTTP-Clients wie Axios Requests ausführen und Authentifizierungsinformationen in Form von Tokens (z. B. JWT) injiziert werden.
In der Django-Anwendung wird auf eine solche API-Integrationsschicht verzichtet, da Frontend und Backend innerhalb desselben Systems arbeiten.
Benutzerinteraktionen werden direkt über HTTP-GET- und POST-Requests an Django Views übermittelt, ohne zusätzliche Service- oder Client-Abstraktionen.
Authentifizierung, Autorisierung und Datenverarbeitung erfolgen vollständig serverseitig, wodurch die Architektur vereinfacht und die Angriffsfläche reduziert wird.

Quellenverzeichnis

ChatGPT, Copilot zur Code generierung (hauptsächlich HTML templates) und Architektur Diskusion/ finden von Dokumentationen

https://docs.djangoproject.com/en/6.0/

https://htmx.org/

https://www.youtube.com/@NetNinja

https://v2.tailwindcss.com/docs