MedPlanner

Web-Anwendungsentwicklung Sommersemester 2021

Egidia Cenko
Medieninformatik
e.cenko@oth-aw.de

Madina Kamalova
Medieninformatik
m.kamalova@oth-aw.de

Matthias Schön
Medieninformatik
m.schoen@oth-aw.de

Christoph Schuster
Medieninformatik
c.schuster1@oth-aw.de

Andrei Trukhin
Medieninformatik
a.trukhin@oth-aw.de

***Zusammenfassung*—Beschreibung der Software-Architektur für das Projekt *MedPlanner* von Team Grün. Dabei wird zunächst allgemein ein Überblick über das System gegeben werden allgemeine Anforderungen an das System beschrieben und eine**

***Index Terms*—TODO oder ganz weg**

I. Überblick

A. Mission Statement

MedPlanner bietet die Möglichkeit, ärztliche Termine übersichtlich zu verwalten. Es handelt sich hierbei um eine Web-Anwendung, die gezielt auf das Selbstmanagement von Arztterminen abgestimmt ist. MedPlanner ist auf verschiedenen Geräten, einschließlich Computern, Smartphones und Tablets verfügbar. Mithilfe von MedPlanner können zukünftige Arzttermine eingetragen und geplant werden. Es ist vor allem für Privatpersonen gedacht, welche somit einen Überblick über die Vielzahl ärztlicher Untersuchungen behalten können.
Zu den wesentlichen Features gehört die Eintragung von Terminen, die man zusätzlich mit Notizen und Tags versehen kann. Weiterhin ist es möglich, Kontaktinformationen für die eigenen Ärzte abzuspeichern. So ist durch MedPlanner geboten Kontaktinformationen, wie zum Beispiel Telefonnummer, Adresse und, falls vorhanden, die Webseite der Arztpraxis, abzurufen, ohne extra vor einer Terminvereinbarung wiederholt nach den nötigen Informationen zu suchen. MedPlanner bietet außerdem die Funktion Erinnerungs-Mails für vereinbarte Termine zu erhalten. So bekommt der Patient direkt nach Termineintragung in der Web-Anwendung eine EMail-Benachrichtung mit den wichtigsten Informationen zum Arzttermin.

B. Architekturziele

Tabelle I
Qualitätsanforderungen

Ziel	Erklärung
Benutzbarkeit	Intuitive Bedienbarkeit und schnelle Erlernbarkeit
Sicherheit	Inhalte sind vor unberechtigtem Zugriff geschützt
Wartbarkeit	Leichte Erweiterbarkeit und Änderung
Leicht zu betreiben	Die Anwendung kann ohne größere Anpassungen genutzt werden

C. Kontextabgrenzung

Die folgende Darstellung visualisiert das zu beschreibende System als Blackbox und zeigt so die Interaktion mit Benutzern und Fremdsystemen auf.

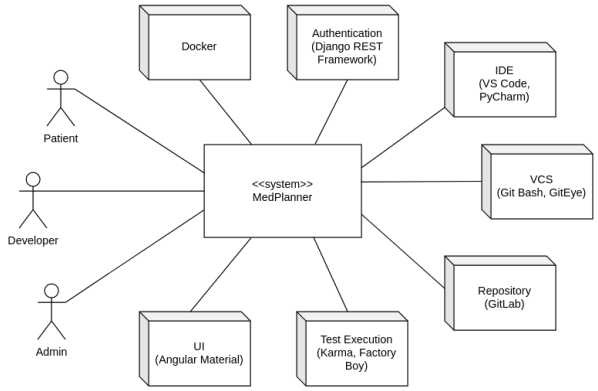

Abbildung 1. Systemkontext für MedPlanner

Menschlicher Akteur ist neben dem Entwickler der Web-Anwendung auch der Admin, der u.a. Daten in die Datenbank einfügen kann, die den Patienten zur Verfügung stehen, wie z.B. die Fachrichtung von Ärzten. Der Patient ist hierbei der eigentliche Nutzer der Web-Anwendung, der eigene Termine und Ärzte einfügen kann. Als externe Systeme gelten neben der Entwicklung und Versionsverwaltung auch weitere Systeme zur Realisierung der Web-Anwendung, z.B. wird für das User Interface *Angular Material* eingesetzt, welches Design-Komponenten bereitstellt. Für die Erstellung von Test wird im Frontend *Karma* und im Backend *Factory Boy* genutzt.

II. Lösungsstrategie

A. Lösungsansätze

Die vorher benannten Qualitätsziele wurden bei MedPlanner folgendermaßen berücksichtigt.

- **Benutzbarkeit**:
  - Intuitives, modernes User Interface
  - Filterung von Terminen für eine erhöhte Übersichtlichkeit der Informationen

- **Sicherheit**:
  - zustandslose Authentifizierung mittels Tokens
  - Passwortspeicherung in Form eines Hashwertes
- **Wartbarkeit**:
  - Modulare Implementierung in Python
  - Style-Komponenten und Farbwahl im Frontend schnell anpassbar
  - Microservice-Architektur unter Verwendung von Docker
- **Leicht zu betreiben**:
  - üblicher Web-Browser als Client genügt
  - lokale Speicherung dem RDBMS [1] SQLite

B. Technologie-Stack

Im Frontend nutzt Medplanner Angular, im Backend das Framework Django in Kombination mit Django REST, um WEB-APIs für das Frontend bereitzustellen und somit eine Verknüpfung zwischen Django und Angular zu ermöglichen. Für die Speicherung der Daten wird die relationale Datenbank SQLite verwendet.

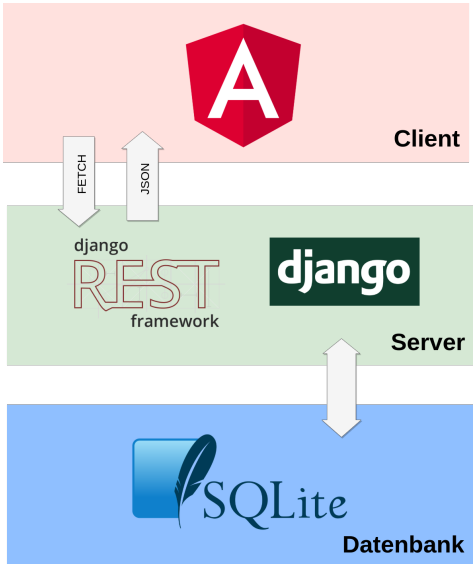

Abbildung 2. Verwendeter Technologie-Stack für MedPlanner

C. Architekturentscheidungen

Wie bereits vorher genannt, wird im Frontend das Framework Angular verwendet. Es stellt einen clientseitigen MVC dar, wodurch eine bessere User Experience mit kürzeren Reaktionszeiten ermöglicht wird. Als Programmiersprache wird TypeScript genutzt, was im Gegensatz zu JavaScript eine Typisierung zulässt und somit logische Typfehler beseitigt. Ein weiterer Grund für diese Entscheidung war die Modularität und Wiederverwendbarkeit. In Angular können Komponenten erstellt und in Module gruppiert werden, die an beliebigen Stellen wiederverwertbar sind. Für einzelne Komponenten ist zudem die Testabdeckung möglich.
Für das Backend fiel die Entscheidung auf Django, da es in der Programmiersprache Python implementiert wird, welche vom gesamten Team gut beherrscht wird. Außerdem bietet Django einige Features an wie beispielsweise die Benutzerauthentifizierung oder die Administration der Inhalte über ein Interface. Mithilfe des Django REST Frameworks (DRF) können APIs in Django zur Verfügung gestellt werden, um den Datenaustausch zwischen Angular und Django zu realisieren.
Für die Abspeicherung der Daten im Backend wird die relationale Datenbank SQLite verwendet, da die Einrichtung unkompliziert abläuft und die Datenbank trotz geringer Größe viele Funktionalitäten bietet [1], die ebenfalls in mächtigeren SQL-Datenbank-Managementsystemen wie MySQL enthalten sind. Im Zusammenhang mit RDBMS und Django ist zu erwähnen, dass das Python-Framework die Realisierung von *n-n-Beziehungen* [2] erlaubt. Dadurch müssen die Entiäten nicht so überarbeitet werden, dass jeweils weitere Tabellen existieren, um one-to-many-Beziehungen zu gewährleisten. Vorteil von Djangos Umsetzung ist, dass in der Datenbank diese Attribute für eine Entität bereits als Liste abgespeichert werden, wodurch größere Anpassungen für die Nutzung und Darstellung im Frontend wegfallen.

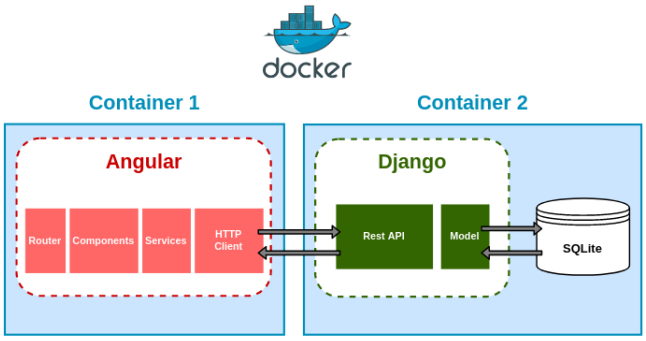

Abbildung 3. Überblick über das System

Für die Realisierung der Web-Anwendung wird eine Microservice-Architektur unter Verwendung von Docker genutzt. Dabei wird jeweils das Frontend sowie das Backend zusammen mit der Datenbank als Container entwickelt und bereitgestellt. Mittels Docker Compose werden die Container in einem Netzwerk verbunden und können so miteinander kommunizieren. Der Vorteil hierbei ist die schnelle Adaption neuer technologischer Trends: Das System kann ohne unnötigen Aufwand durch eine entsprechende Komposition auch mit anderen Webcontainer eingesetzt werden.

[1] Abk. für Relational Database Management System

[2] Auch bekannt als many-to-many

D. Struktur

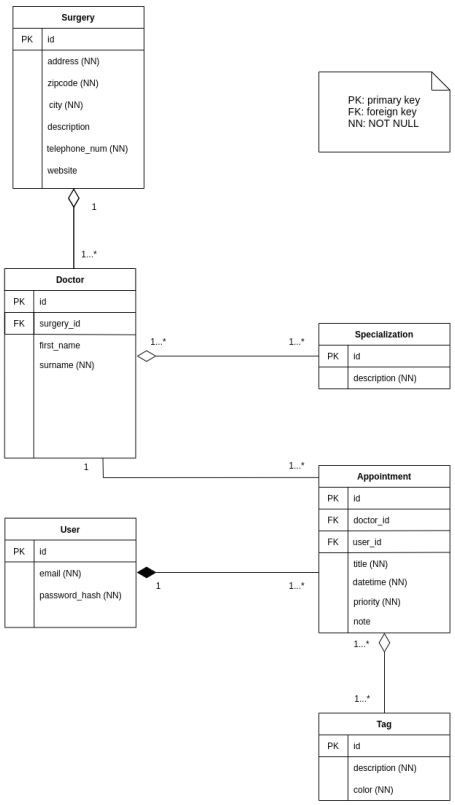

Abbildung 4. Entity-Relationship-Modell

Django models repräsentieren in der Datenbank die unterschiedlichen Entitäten. Dabei wird vom Framework bereits standardmäßig ein User model bereitgestellt, was beispielsweise Attribute wie Vor- und Nachname enthält. [2] Da in MedPlanner neben der EMail-Adresse und dem Password zu jetzigem Zeitpunkt keine weiteren Informationen benötigt werden, wurde ein angepasstes User model erstellt, was auf die Anforderungen von MedPlanner abgestimmt ist.

Für die Identifizierung der Benutzer wird bei MedPlanner die *tokenbasierte Authentifizierung* eingesetzt: So wird bei jeder Registrierung bzw. bei jedem Login ein neues Token für den Benutzer erstellt, womit bestimmte Quellen ohne zusätzliche EMail- oder Passwortinformation abrufbar sind, so beispielsweise auch das Bearbeiten von Terminen. Allgemein werden die Tokens überall dort eingesetzt, wo eine Eingabe von EMail und Passwort nicht dringend benötigt werden.

III. Risiken und Probleme

Im Abschnitt II-D wurde bereits die Anpassung des User models erwähnt. Standardmäßig wird für das Einfügen eines neuen Benutzers ein Benutzername abgefragt. Durch die Überschreibung des User models wird jedoch für eine Registrierung neben dem Passwort nur eine EMail-Adresse verlangt. Damit dieses Verhalten korrekt übernommen wird, muss zudem eine Überschreibung des Admin models vorgenommen werden, da sonst Konflikte entstehen im Backend entstehen. Nach dieser Abänderung ist eine Neuerzeugung der Datenbank erforderlich, um die Aktualisierungen einzupflegen. Aus diesem Grund ist es zwingend ratsam derartige Anpassungen in der Entwicklungsphase vorzunehmen, um größere Migrationsprobleme oder das Risiko für den Verlust von Nutzerdaten zu vermeiden.
TODO: Wo sind zu wenig Tests, wodurch das System vielleicht abstürzt? Mögliche schädliche Ereignisse, die Einfluss auf die Softwarearchitektur haben (oder hatten) (als Auflistung)

IV. Fazit und Ausblick

Zum aktuellen Stand bietet MedPlanner ein solides Selbstmanagement-Tool für die Verwaltung von Arztterminen sowie Kontaktinformationen für Arztpraxen. Es ist zudem möglich, direkt eine EMail-Benachrichtigung zu erhalten, sobald ein Patient einen neuen Termin bei MedPlanner erstellt. Für die Zukunft kann diese Benachrichtigungs-Funktionalität so erweitert werden, dass eine Benachrichtigung 24 Stunden vor Terminstart gesendet wird.
Außerdem ist denkbar, dass sich Patienten neben geplanten Terminen in Zukunft auch Reminder in einer bestimmten Erinnerungs-Periode setzen zu können, um an das Vereinbaren eines Termins für Routineuntersuchungen erinnert zu werden. In zukünftigen Versionen könnte MedPlanner auch so ausgebaut werden, dass Patienten neben der Terminverwaltung auch ihre Arztbefunde sicher abspeichern können, um somit alle gesundheitlichen Angelegenheiten kompakt in einer Web-Anwendung strukturieren zu können.

Welche optionalen Features sind noch zu realisieren? Wie könnte das System zukünftig weiter ausgebaut werden? (z.B für Ärzte)

Literatur

[1] https://sqlite.org/features.html
[2] https://docs.djangoproject.com/en/3.2/ref/contrib/auth/#django.contrib.auth.models.User