

Szegedi Tudományegyetem
Informatikai Intézet

**Fogászati rendelő webalkalmazás
implementálása Django keretrendszerben**

Szakedolgozat

Készítette:

Sándor Márton

programtervező informatikus BSc
szakos hallgató

Témavezető:

Antal Gábor

adjunktus

Szeged

2025

Feladatkiírás

A szakdolgozat célja, hogy a hallgató megismerkedjen az egyik legismertebb Python alapú webes keretrendszerrel, a Django-val. A megszerzett ismeretei segítségével a hallgató egy fogászati szakrendelőnek tervezett időpontfoglalási rendszert épít, amelyben elérhető a PayPal fiókos, és a bankkártyás fizetés, és az időpontfoglalás. A rendelő orvosainak lehetőségük van a saját munkaidejük bevitelére, a páciensek kezeléstörténetének megtekintésére, és a magukhoz foglalt időpontok megtekintésére. Továbbá bármelyik páciens adatait és elérhetőségeit megtekinthetik. A rendszer emellett küld email-es értesítéseket is, hogy a felhasználók és az orvosok is kapjanak információkat a foglalásokról. Az alkalmazás tartalmaz admin felületet is, amivel a felhasználónak joga van új orvosokat hozzáadni a rendszerhez, a felhasználóknak új jelszót adni, és az adatbázisban tárolt kezeléseket létrehozni, szerkeszteni, vagy törölni. A felhasznált adatbázis kezelő rendszer a Django beépített rendszere, ami sqlite3 adatbázist használ.

Tartalmi összefoglaló

A téma megnevezése:

Fogászati rendelő webalkalmazás implementálása Django keretrendszerben

A megadott feladat megfogalmazása:

A Django keretrendszer által nyújtott előnyök megismerése, felhasználása a fogászati rendelő időpontfoglalási rendszerének fejlesztéséhez. Az alkalmazásnak rendelkeznie kell felhasználói, orvosi, és adminisztrátori felülettel. Meg kell valósítani az online időpontfoglalás, és az online fizetés lehetőségét. Az orvosoknak meg kell valósítani egy felületet amivel vissza tudják nézni a páciens kezeléstörténetét. Az adatokat pedig FHIR szabvány szerint kell tárolni, hogy bármilyen más egészségügyi rendszerrel kompatibilis legyen.[2]

A megoldási mód:

Megismertem a Django keretrendszerrel, annak felépítésével. Megterveztem a rendelő webalkalmazásának adatbázis-struktúráját, amely tartalmazza a páciensek, orvosok, kezelések és időpontfoglalások moduljait. Az alkalmazás reszponszív felhasználói felületét modern CSS megoldásokkal építettem ki, míg a backend részben a Django admin és REST API-k biztosítják az adatok hatékony kezelését. Emellett integráltam a PayPal fizetési rendszert az online fizetések lebonyolításához. A fejlesztés során a projektet GitHubon verzióztam, biztosítva ezzel a kód stabilitását és könnyű karbantarthatóságát.

Alkalmazott eszközök, módszerek:

Git, Github, Django, SQLite, HTML, JavaScript, CSS, Python

Elért eredmények:

Megismertem a Django-t, és az általa nyújtott lehetőségeket, a PayPal integrációt, a Python nyelvet, az FHIR szabványt, és a webfejlesztést. Sikeresen működik az általam integrált időpontfoglalási és fizetési rendszer.[4]

Kulcsszavak:

Django, SQLite, HTML, JavaScript, CSS, Python, ORM

Tartalomjegyzék

Feladatkiírás	1
Tartalmi összefoglaló	2
Tartalomjegyzék	4
1. A Django keretrendszer	6
1.1. Az MVT programszervezési minta	6
1.2. Model	6
1.3. View	7
1.4. Template	7
2. Alkalmazás struktúrája	8
3. Adatbázis	10
3.1. Az adatbázis típusa	10
3.2. Az adatbázis felépítése	11
3.2.1. Profilkezelés	12
3.2.2. Az időpontfoglalás adatainak tárolása	13
3.2.3. Az orvosok munkaidejének tárolása	14
4. A Template struktúra	16
5. User-szintű felhasználói felületek, és azok működése	18
5.1. A bejelentkezés és a regisztráció működése	18
5.1.1. Az alkalmazás autentikációra használt metódusai:	18
5.1.2. Regisztráció	19
5.1.3. Bejelentkezés	20

5.2. A kezdőoldal	21
5.3. Az időpontfoglalás	21
5.3.1. Az időpontfoglalás lépései	22
5.4. A fizetési oldal	25
5.5. A profil oldal	27
6. Staff-szintű felhasználói felületek, és azok működése	29
6.1. A "Páciensek" oldal	29
6.2. Az "Időpontjaim mára" oldal	30
6.3. A "Munkaidő oldal"	31
6.4. Az orvosok Profil oldala	32
7. Superuser-szintű felhasználói felületek, és azok működése	35
7.1. Az "Admin" oldal	35
8. Design	39
9. Összefoglaló	42
Nyilatkozat	44
Irodalomjegyzék	45

1. fejezet

A Django keretrendszer

A Django egy magas szintű Python webkeretrendszer, amely támogatja a gyors fejlesztést és az egyszerű, jól átgondolt megoldásokat. Tapasztalt fejlesztők által készített, így számos webfejlesztési nehézséget megold, és lehetővé teszi, hogy a fejlesztő alkalmazás írására koncentráljon, anélkül, hogy újra fel kellene találnia a kereket. További pozitívuma, hogy ingyenes, és nyílt forráskódú.[1]

1.1. Az MVT programszervezési minta

Django projekt lévén az alkalmazás az MVT (Model View Template) design pattern alapelveit kell, hogy kövesse. Ez áll a modellből, ahol az adatbázis struktúráját építjük fel, a view-ből, ami lényegében a projekt azon része, ahol a háttérfolyamatok futnak, és a template-ből, ami a felhasználói felületet tartalmazza. Ez a kapcsolata a felhasználónak az alkalmazással.

1.2. Model

A modellek a Django alkalmazáson belüli adatszerkezet kezelését és interakcióját irányítják, így a Django alkalmazások alapját képezik, mivel az adatok kritikus szerepet játszanak. A Django Modellek egy erőteljes, Objektum-Relációs Leképezést (ORM: Object-Relational Mapping) megvalósító funkciót használnak, amely áthidalja a szakadékot a relációs adatbázis és a Python kód között. Ez a leképezés a Python objektumokat (osz-

tályokat) adatbázis táblákká alakítja, az osztályok attribútumait oszlopokká, és az egyes példányokat a táblák soraivá. Az ORM egyik nagy előnye, hogy lehetővé teszi az adatbázissal való interakciót Python objektumokon keresztül, anélkül, hogy SQL lekérdezéseket kellene írunk. A Django modellek összegzik az összes adatbázissal kapcsolatos logikát és meghatározzák az adatbázis szerkezetét, mint egy tervrajzot annak, hogy milyen adatokat szeretnénk tárolni.[3]

1.3. View

Ha az MVC modellhez szeretnénk hasonlítani, akkor az MVT modellben a View hasonló, mint az MVC-ben a Controller. A Django view-k felelősek a felhasználói kérések feldolgozásáért és a válaszok visszaküldéséért. Híd szerepét töltik be a Model és a Template között: Adatokat gyűjtenek a modellből, logikai műveleteket (például bizonyos kritériumok alapján végzett lekérdezéseket) hajtanak végre rajtuk, majd az eredményeket átadják a Template-nek a megjelenítéshez. A View-kat függvényekként vagy osztály alapú View-ként is megírhatjuk, attól függően, hogy az alkalmazásunk komplexitása és követelményei melyik megközelítést igénylik.[3]

1.4. Template

A Django Template-ek feladata, hogy a böngészőben megjelenítendő végső HTML kimenetet rendereljék. Meghatározzák, miként kell az adatokat bemutatni, HTML és a Django sablonnyelvének kombinációjával. A Django sablonnyelv template tageket (`{% %}`) és template változókat (`{{ }}`) alkalmaz, amelyek lehetővé teszik, hogy a sablon HTML kódjában Django módba lépjen, és így hozzáférjen a View-kban definiált változókhoz, illetve vezérlési struktúrákat használjon a megjelenítés szabályozására. A sablonok továbbá formázhatók CSS-sel, illetve bármely kedvelt CSS keretrendszerrel, hogy a felhasználói felület még szebb legyen. Emellett animálhatók is JS segítségével.[3]

2. fejezet

Alkalmazás struktúrája

Egy Django projekt esetében a projekt felépítése modulárisan, egy vagy több alkalmazásból (app) áll, melyek mindegyike egy adott funkcionális területért felel. A szakdolgozatom esetében a "rendelo" mappa tartalmazza a teljes webalkalmazás forráskódját.

A "rendelo" mappa a következő részekből áll:

- Gyökérszint:
 - manage.py: A Django projekt parancssori kezelője, amely a fejlesztési feladatok (például migrációk futtatása, szerver indítása) végrehajtását segíti.
 - db.sqlite3: Az alapértelmezett, fejlesztési környezetben használt SQLite adatbázis fájlja.
- Projekt főkönyvtára ("rendelo"): Itt találhatóak a projekt globális beállításait és konfigurációs fájljait, mint például a settings.py, urls.py, wsgi.py és asgi.py. Ezek a fájlok felelősek az alkalmazás működésének alapvető paramétereinek meghatározásáért, az útvonalak kezeléséért és a szerverrel való kommunikációért.
- Alkalmazás könyvtára ("rendeloweboldal"): Ez a rész tartalmazza a rendszer egyes moduljait, amelyek a következő fő komponensekből állnak:
 - models.py: Az adatbázis szerkezetét definiáló modellek, melyek meghatározzák a páciensek, orvosok, kezelések, időpontfoglalások és az időpont foglalások fizetési státuszának struktúráját.

- views.py: A felhasználói kérések feldolgozásáért és az üzleti logika megvalósításáért felelős fájl, amely összeköti a modelleket a sablonokkal.
- forms.py: Az űrlapok és azok validációs szabályainak definíciója, melyek révén az adatbevitel és ellenőrzés történik.
- urls.py: Az alkalmazás specifikus URL-konfigurációja, amely a különböző view-k elérését biztosítja.
- admin.py: A Django beépített adminisztrátori felület konfigurációját tartalmazza. Hozzá kell adni az összes modellt az adatbázisból, amit elérhetővé szeretnénk tenni rajta.
- migrations/: Az adatbázis változásait követő migrációs fájlokat tartalmazza, dokumentálva a modellek módosításait.
- static/ és templates/: A statikus fájlokat (CSS, JavaScript, és az alkalmazás designjához tartozó képek) illetve a HTML template-eket rendszerezi, amelyek a felhasználói felület megjelenítéséért felelősek.

A projekt kialakítása moduláris és átlátható, mely lehetővé teszi a fejlesztés, karbantartás és bővítés egyszerű kezelését. Emellett a projekt verziókezelése a GitHubon történik, így könnyen nyomon követhető az egész alkalmazás fejlesztése.

3. fejezet

Adatbázis

3.1. Az adatbázis típusa

Először is el kell döntenünk, hogy milyen adatbázist fogunk használni a fejleszteni kívánt alkalmazásunkhoz. Ennek sok szempontja lehet, például hogy mekkora mennyiségű adattal szeretnénk dolgozni, vagy hogy mennyire kell gyorsnak lennie a lekérdezéseknek.

3.1. ábra. Példa az adatbázis beállításra

```
76  DATABASES = {  
77      'default': {  
78          'ENGINE': 'django.db.backends.sqlite3',  
79          'NAME': BASE_DIR / 'db.sqlite3',  
80      }  
81  }
```

A szakdolgozatomban a Django beépített adatbázisát használtam, ami sqlite3, mivel egy fogászati rendelőnek nincsen nagyon nagy adatforgalma, így elegendő hozzá ez a fajta adatbázis. A 3.1 ábrán látható egy példa az adatbázis típusának beállítására. Ezt a beállítást a settings.py fájlban kell megadni, ami az én alkalmazásomban a projekt fő mappáján belüli "rendelo" nevű mappában található.[5]

3.2. Az adatbázis felépítése

Az adatbázis típusának kiválasztása után a legfontosabb rész következik: Felépíteni az adatbázis szerkezetét. Mivel ORM technológiát használ a keretrendszerünk így szerencsére nincs szükségünk SQL ismeretekre ennek a műveletnek a végrehajtásához. Django-ban minden alkalmazásnak van egy (vagy több) `models.py` nevű fájlja. Ebbe importálnunk kell a Django `models` modulját, amit így tehetünk meg:

```
from django.db import models
```

Ezután Python osztályként beleírhatjuk a fájlba a tárolni kívánt adatok tulajdonságait. Az ORM-ben az adatbázis táblák oszlopait a Python osztályaink adják meg, és az adatbázis rekordok ezeknek a példányaiból keletkeznek. Minden modell osztálynak közvetlenül, vagy közvetetten a `models.Model` modulból kell származnia.

3.2. ábra. Példa az adatbázis modell leírására

```
26 class Doctor(models.Model):
27     id = models.CharField(max_length=64, primary_key=True)
28     name = models.CharField(max_length=255)
29     photo = models.ImageField(upload_to='doctor_pictures/', null=True, blank=True)
30     qualification = models.TextField(null=True, blank=True)
31
32     def __str__(self):
33         return self.name
```

A szakdolgozatomban egyértelmű volt, hogy FHIR szabványú adatbázissal kell dolgoznom, mivel az egészségügyi alkalmazásoknak ez a szabványa. Azért éri meg így kialakítani az adatbázist, mert ezzel a módszerrel az összes egészségügyi rendszerrel kompatibilis rendszert hozhatunk létre. A 3.2. ábrán látható az egyik FHIR szabványú modell osztály a szakdolgozatomból.

A modellek megírása nem volt egyszerű, mivel meg kellett oldanom a profilkezelést és az autentikációt az alkalmazásban, viszont szabványosnak kellett maradnia. Ezzel az a probléma, hogy az FHIR szabvány nem támogat profilkezelést.

Az alkalmazás modell osztályai:

- RendeloUser
- Patient
- Doctor

- Treatment
- Appointment
- WorkingHours
- PaymentStatus

3.2.1. Profilkezelés

3.3. ábra. A felhasználói profilokat tároló modell

```
6 class RendeloUser(AbstractUser):
7     id = models.CharField(max_length=64, primary_key=True, default=str(uuid4()))
8     email = models.EmailField(unique=True)
9
10    USERNAME_FIELD = 'email'
11    REQUIRED_FIELDS = ['username']
12
13    def __str__(self):
14        return self.email
```

Az alkalmazásomban három felhasználói jogosultságú profil elérhető:

- Superuser: Adminisztrátor akinek mindenhez van joga, bármilyen adatot törölhet, meg van valósítva a számára egy külön "Admin" nevű oldal amin az adatok kezeléséhez hozzáfér, és beléphet a Django beépített adminisztrátori felületére is.
- Staff: A fogászati rendelő orvosainak a jogosultságával rendelkeznek. Ezeket a profilekat az adminisztrátor tudja létrehozni az "Admin" oldalon. Minden ilyen profil megjelenik orvosként az orvosok kiválasztásánál az időpontfoglalás során. Joga van megtekinteni a páciensek kezeléstörténetét, és szerkeszteni azt. Joga van az összes hozzá foglalt időpont megtekintésére, a páciensek adatainak lekérdezésére, és a saját munkaidejének a beállítására.
- User: A páciensek felhasználói fiókjai, a Regisztrációs oldalon hozhatók létre. Az időpontfoglaláshoz csak ezeknek a fiókoknak van joga, emellett megtekinthetik a saját kezeléstörténetüket és az időpontjaikat, illetve le is mondhatják azokat.

A profilokat a 3.3. ábrán látható `RendeloUser` osztály tárolja. Az osztály az `AbstractUser`-ből származik, ami a Django beépített autentikációs rendszerébe tartozik. Amikor létrehozunk az alkalmazásban egy bármilyen jogosultsággal rendelkező profilt, akkor ez az osztály fogja tárolni a bejelentkezéshez szükséges adatainkat. Superuser esetén csak ez az egy objektum tartozik a profilunkhoz, mivel az adminisztrátor nem orvos, és nem páciens, nincs szüksége további adat tárolására.

Ha az adminisztrátor létrehoz egy új orvosnak egy profilt az alkalmazásban, akkor először is létrejön a `RendeloUser` példány, amihez generálódik egy uuid4 egyedi azonosító, ami `id` néven van tárolva a 3.3. ábrán látható módon. Ezután létrejön egy FHIR szabványos `Doctor` osztálypéldány is, ami az orvos adatait tárolja. Ennek az osztálynak is van egy `id` nevű, karaktersorozat típusú adattagja, amibe bemásolja a program a `RendeloUser` objektumban létrehozott egyedi azonosítót. Így kapcsolódik össze a két osztálypéldány. Ezen kívül a program hozzáadja a többi tárolni kívánt adatát az orvosnak.

Hasonló folyamat történik, amikor egy user szintű felhasználó regisztrál az alkalmazásba. Először létrejön a `RendeloUser` példány az egyedi uuid4 azonosítóval, ezután létrejön egy FHIR szabványos `Patient` objektum, amiben a páciens adatait tároljuk, a `Patient` objektum `id` nevű karaktersorozat típusú adattagjába pedig bemásolja a program az egyedi azonosítót, így összekapcsolta a profiladatokat tároló `RendeloUser` példányt a páciens adatokat tároló `Patient` példánnyal. Ezek után pedig elmenti a user további adatait is. A 3.4. ábrán látható a `Patient` modell.

3.4. ábra. A `Patient` modell

```
16 class Patient(models.Model):
17     id = models.CharField(max_length=64, primary_key=True)
18     name = models.CharField(max_length=255)
19     gender = models.CharField(max_length=10, choices=[('male', 'Male'), ('female', 'Female')])
20     birthDate = models.DateField(null=True, blank=True)
21     telecom = models.CharField(max_length=255, null=True, blank=True)
22
23     def __str__(self):
24         return self.name
```

3.2.2. Az időpontfoglalás adatainak tárolása

Az időpontfoglalás adatait három modellben tárolja az alkalmazás:

- Treatment modell: A kezelés típusát tárolja, amire időpontokat lehet foglalni. A kezeléseket az adminisztrátor viszi fel az adatbázisba. A `Treatment`-nek is van egy

id adattagja amibe generálódik automatikusan az uuid4, ezen kívül tárol egy nevet, leírást, egy hosszt, ami azt tárolja, hogy mennyi ideig tart, illetve egy árat ami forintban értendő. Ez a modell is FHIR szabványt követ.

- Appointment modell: A lefoglalt időpontot tárolja. Ez az osztálypéldány a páciensek időpontfoglalásának a hatására jön létre. Tárolja a saját id-ját ami automatikusan generálódik a számára amikor létrejön, ezen kívül van egy *patient* adattagja ami az időpontot lefoglaló páciens id-ját tárolja. A *practitioner* adattag ahhoz az orvoshoz kapcsolódik ForeignKey-ként, akihez a páciens időpontot foglalt. Erre azért volt szükség, hogy ha töröljük az orvost az adatbázisból, akkor a hozzá foglalt időpontok is törlődjenek automatikusan. A *treatment* is hasonló, de a lefoglalt kezeléshez kapcsolódik ForeignKey-ként. A start és end adattagok `models.DateTimeField` típusúak, és az időpont kezdetét, és végét tárolják. A *status* karaktersorozat típusú adattag az időpont státuszát tárolja, ami ha "foglalt", akkor nem lehet rá időpontot foglalni. A *custom_description* egy `TextField` típusú adattag, ami a kezeléstörténet tárolására szolgál. Ez az adattag a példány létrehozásakor üres, és a kezelést végző orvos tudja szerkeszteni, a páciensnek csak a megtekintéséhez van joga. Ez a modell is FHIR szabványt követ.
- PaymentStatus modell: Ez már nem FHIR szabvány szerinti, viszont szükséges, mert ez tárolja a lefoglalt időpont fizetési státuszát. Tartalmaz egy *appointment* nevű, `models.OneToOneField` típusú adattagot, ami által az időponthoz kapcsolódik. (Időpont foglalás esetén automatikusan jön létre az osztálypéldány) Emellett tartalmaz egy `bool` típusú adattagot aminek *is_paid* a neve, és azt tárolja hogy a felhasználó kifizette-e az időpontjára lefoglalt kezelés árát. Ha "true" az értéke, akkor kifizette, ha "false", akkor nem.

3.2.3. Az orvosok munkaidejének tárolása

Az orvosok munkaidejének tárolása a `WorkingHours` modellben történik. Az orvos a "working_hours.html" oldalon kiválaszthat egy dátumot, és arra a napra beállíthatja a saját munkaidejét. A modell tartalmaz egy *doctor* nevű, `models.ForeignKey` típusú adattagot, ami által hozzá kapcsolódik az orvosnak létrejött `Doctor` osztálypéldányhoz.

Van egy *date* adattagja, amiben azt a dátumot tárolja, amelyikre a munkaidő be lett állítva. Ezek mellett van egy *start*, és egy *end* adattag, amik értelem szerűen a munkaidő kezdési, és befejezési időpontját tárolják. Ez a modell fontos, mert a páciensek számára megjelenő elérhető időpontok ez alapján jelennek meg.

4. fejezet

A Template struktúra

Egy webes alkalmazásnak a Template a leglátványosabb része, hiszen ez jeleníti meg a felhasználó számára a kezelőfelületet és animációkat. A Django támogatja a HTML Template-ek öröklődését, ami azért hasznos, mert ezzel sokkal átláthatóbb, és könnyebben karbantartható webalkalmazást kapunk. Az öröklődéshez először is szükség van egy "fő" HTML oldalra, amiből származtatni fogjuk a többi. Eerre az oldalra importálhatjuk a CSS és JS fájlokat, amik az egész alkalmazás stílusát és animációit állítják be. Továbbá a navigációs sávot, a fejléct és a lábléct is itt érdemes megadni. Amikor a fő oldallal készen vagyunk akkor származtathatunk belőle, és minden származtatott oldal megkapja a szülő oldal tulajdonságait. A származtatást úgy oldhatjuk meg, hogy a származtatott oldal elejére az alábbi kódot írjuk:

```
{% extends "base.html" %}
```

Ezzel az új html oldalunk szülője a base.html lett. A szakdolgozatomban a base.html oldalt készítettem el a "fő" HTML fájlak. Ebbe írtam meg a fejléct, az importokat, a navigációs sávot, és a lábléct. Meg kellett adnom azokat a részeket az oldalon, amiket a belőle származó HTML oldalak változtatni fognak.

4.1. ábra. Példa az öröklődésben használt block-okra

```
74      <main>
75      |      {% block content %}{% endblock %}
76      </main>
```

Az 4.1. ábrán egy példa látható az oldal main részének a block-jára. Ezután, ha írni szeretnénk egy származtatott oldalon ebbe a block-ba, akkor a "content" nevű block-ba szánt kódot `{% block content %}` és `{% endblock %}` kód között kell megadnunk. Ennek hatására a származtatott oldalon megadott block kódot a base.html oldal main részébe fogja helyezni.

4.2. ábra. A navigációs sáv kódja

```
49 <nav>
50   <ul class="navbar">
51     <li><a href="{% url 'home' %}">Kezdőoldal</a></li>
52     <li><a href="{% url 'book' %}">Időpontfoglalás</a></li>
53     {% if user.is_authenticated and user.is_superuser %}
54     <li><a href="{% url 'admin_view' %}">Admin</a></li>
55     {% endif %}
56     {% if user.is_authenticated and user.is_staff and not user.is_superuser %}
57     <li><a href="{% url 'patients' %}">Páciensek</a></li>
58     <li><a href="{% url 'appointments_today' %}">Időpontjaim mára</a></li>
59     <li><a href="{% url 'working_hours' %}">Munkaidő</a></li>
60     {% endif %}
61     {% if user.is_authenticated %}
62     <li><a href="{% url 'profile' %}">Profil</a></li>
63     <li><a href="{% url 'logout' %}">Kijelentkezés</a></li>
64     {% else %}
65     <li><a href="{% url 'login' %}">Bejelentkezés</a></li>
66     {% endif %}
67     <li class="icon">
68       <button onclick="toggleColorScheme()">
69         
72     </li>
73   </ul>
74 </nav>
```

A navigációs sáv kulcsfontosságú szerepet tölt be az alkalmazás felhasználhatóságában. A szakdolgozatomban az 4.2. ábrán látható módon oldottam meg a navigációs sáv implementációját a base.html fájlban. Az urls.py fájlban található linkeket adtam hozzá a Django sablonnyelvben írt feltételekkel, hogy a különböző jogokkal rendelkező felhasználók csak a nekik szánt oldalakat láthassák rajta. Ezek a linkek az alkalmazás további felhasználói felületeire navigálják a felhasználót. Ezekén felül pedig van egy gomb is a navigációs sávon, ami a sötét, és világos módok közötti váltást teszi lehetővé egy JS kód segítségével, amire még a későbbiekben kitérek.

5. fejezet

User-szintű felhasználói felületek, és azok működése

A fogászati rendelő alkalmazás számos felülettel rendelkezik.

5.1. A bejelentkezés és a regisztráció működése

Az alkalmazásomban a regisztráció és bejelentkezés funkciók a Django beépített autentikációs rendszerére épülnek, amely lehetővé teszi a felhasználók kezelését, hitelesítését és jogosultságainak kezelését. Ez a funkció azért hasznos egy fejlesztőnek, mert ezáltal nem kell ezeket implementálnia minden új projektjébe.

5.1.1. Az alkalmazás autentikációra használt metódusai:

- "authenticate": Ez a metódus ellenőrzi a felhasználó hitelesítő adatait (email és jelszó). Az EmailBackend osztály implementálja az "authentication.py" fájlban, és az email cím alapján keresi meg a felhasználót.
- "auth_login": Bejelentkezteti a felhasználót, és a munkamenethez társítja. A Django beépített metódusa.
- "check_password": Ellenőrzi, hogy a megadott jelszó megegyezik-e a titkosított jelszóval. A Django beépített metódusa.

- "set_password": Beállítja az adott felhasználónak a jelszavát, és hashelt formában elmenti az adatbázisba. A Django beépített metódusa.
- "get_user": Lekéri a felhasználót az azonosítója alapján. A Django beépített metódusa.

5.1. ábra. Az "authentication.py"

```
1  from django.contrib.auth.backends import ModelBackend
2  from .models import RendeloUser
3
4  class EmailBackend(ModelBackend):
5      def authenticate(self, request, username=None, password=None, **kwargs):
6          try:
7              user = RendeloUser.objects.get(email=username)
8              if user.check_password(password):
9                  return user
10             except RendeloUser.DoesNotExist:
11                 return None
12
13             def get_user(self, user_id):
14                 try:
15                     return RendeloUser.objects.get(pk=user_id)
16                 except RendeloUser.DoesNotExist:
17                     return None
```

5.1.2. Regisztráció

A regisztráció során a felhasználó a "register.html" fájlban megvalósított oldalon létrehozhat egy új fiókot, amelyet a rendszer a "RendeloUser" modellben tárol. A regisztrációs folyamat a következőképpen működik:

1. Amikor a felhasználó megnyitja a regisztrációs oldalt, a "register_view" nézet megjeleníti a regisztrációs űrlapot. Az űrlap két részből áll: a "RegistrationForm" a felhasználói fiók alapvető adatait (például felhasználónév, email cím, jelszó) kezeli, míg a "PatientForm" a páciens adatait (például név, születési dátum) tartalmazza.
2. Az űrlap elküldése után a "register_view" nézet ellenőrzi az űrlapok érvényességét. Ha az adatok helyesek, a "RegistrationForm" "save" metódusa létrehoz egy új RendeloUser példányt, amely a felhasználói fiókot reprezentálja. A jelszó titkosítva kerül tárolásra a "set_password" metódus segítségével, ami az "AbstractUser" metódusa, amiből a "RendeloUser" osztály származik.

3. A "PatientForm" létrehoz egy új Patient példányt, amely a páciens adatait tárolja. A "register_view" nézet pedig beállítja a Patient *id* adattagjának az értékét az újonnan létrehozott RendeloUser *id* adattagjának az értékére, hogy a Patient példány azonosítója megegyezzen a RendeloUser példány azonosítójával, így a két objektum összekapcsolódhasson.
4. A regisztráció sikeres befejezése után az "auth_login" metódus automatikusan bejelentkezteti a felhasználót, és átirányítja a kezdőoldalra.
5. A felhasználó beállított email címére egy email kerül kiküldésre, amelyben megköszöni a cég a regisztrációt. (Jelenleg egy konzolos email küldés van beállítva, csupán a demózás céljából.)

5.2. ábra. A register_view nézet

```
352 def register_view(request):
353     if request.method == 'POST':
354         user_form = RegistrationForm(request.POST)
355         patient_form = PatientForm(request.POST)
356         if user_form.is_valid() and patient_form.is_valid():
357             user = user_form.save()
358             patient = patient_form.save(commit=False)
359             patient.id = user.id
360             patient.save()
361             auth_login(request, user, backend='django.contrib.auth.backends.ModelBackend')
362             send_mail(
363                 'Sikeres regisztráció!',
364                 'Kedves {},\n\nSikeresen regisztrált a rendszerünkbe, köszönjük, hogy klinikánkat választotta!\n\nMostantól minden időpon',
365                 'your_email@example.com',
366                 [user.email],
367                 fail_silently=False,
368             )
369             return redirect('/')
370     else:
371         user_form = RegistrationForm()
372         patient_form = PatientForm()
373     return render(request, 'register.html', {'user_form': user_form, 'patient_form': patient_form})
```

5.1.3. Bejelentkezés

A bejelentkezés során a felhasználó a "login.html" fájlban megvalósított oldalon megadja az email címét és jelszavát, amelyek alapján a bejelentkezés történik. A folyamat a következőképpen működik:

1. Amikor a felhasználó megnyitja a bejelentkezési oldalt, a "login_view" nézet megjeleníti a bejelentkezési űrlapot. Az űrlap tartalmazza az email cím és jelszó mezőket.

2. Az űrlap elküldése után a "login_view" nézet az authentication.py "EmailBackend" osztályának "authenticate" metódusát hívja meg, amely az email cím és jelszó páros érvényességét ellenőrzi. Az "authenticate" metódus a "RendeloUser" modellben keresi meg a felhasználót az email cím alapján, majd a "check_password" metódussal ellenőrzi a jelszót.
3. Ha a hitelesítés sikeres, az "auth_login" metódus bejelentkezteti a felhasználót, és a munkamenethez társítja. Ezután a felhasználó átirányításra kerül a kezdőoldalra.
4. Ha a hitelesítés sikertelen (például helytelen email cím vagy jelszó miatt), a rendszer hibaüzenetet jelenít meg a felhasználónak.

5.2. A kezdőoldal

Az összes szintű felhasználó bejelentkezés után a kezdőoldalon találja magát, amit a kezdoldal.html fájlban valósítottam meg. Az oldalon található a base.html elemein kívül egy marketing leírás a rendelőről, ami statikusan az oldalra van írva, nem lehet változtatni, csak a HTML kódban. A leírás után pedig egy táblázat a rendelőben lehetséges kezelésekről, és azok árairól. A táblázat fejléce után Django sablon nyelven következik egy for ciklus, ami végigmegy az összes "Treatment" példányon az adatbázisban, és mindegyiknek a nevét, és az árát kiírja egy külön sorba. A Django Template fájlok az adatbázis objektumait a views.py egyik függvényétől kapják meg az 1.7. ábrán látható módon.

5.3. ábra. A kezdőoldal views.py-ban található megjelenítési függvénye

```
27 def kezdoldal(request):
28     treatments = Treatment.objects.all()
29     return render(request, 'kezdoldal.html', {'treatments': treatments})
```

5.3. Az időpontfoglalás

Az időpontfoglalás oldal az egyik legösszetettebb része az alkalmazásnak. A foglalási folyamat során a felhasználó kiválasztja az orvost, a kezelést, a dátumot és az időpontot,

majd a fizetési módot, ezzel elindítva az időpontfoglalás teljes folyamatát. A `views.py` fájlban definiált `"idopontfoglalas"` függvény koordinálja a folyamatot.

5.3.1. Az időpontfoglalás lépései

1. Orvos kiválasztása:

- **Interakció:** A felhasználó az oldal tetején megjelenített orvosok listájából választ.
- **Kód:** JavaScript eseménykezelő aktiválódik, amely beállítja a `"selected_doctor"` mező értékét a kiválasztott orvos azonosítójára, majd meghívja az `"updateAvailableSlots()"` függvényt.

2. Kezelés kiválasztása:

- **Interakció:** A felhasználó a legördülő menüből választja ki a kívánt kezelést.
- **Kód:** A kezelési opció módosítása szintén az `"updateAvailableSlots()"` függvényt indítja el, frissítve a kezeléshez tartozó időpontokat.

3. Dátum kiválasztása:

- **Interakció:** A felhasználó a naptármezőből választja ki a foglalni kívánt napot.
- **Kód:** A dátumválasztás után a JavaScript lekéri az adott napra vonatkozó szabad időpontokat az API végpontból, majd megjeleníti azokat.

4. Időpont kiválasztása:

- **Interakció:** A megjelenített időpontgombok közül a felhasználó kiválaszt egyet, vagy rákattint a `"legközelebbi időpont"` gombra.
- **Kód:** A kiválasztáskor a JavaScript beállítja az `"appointment_datetime"` mező értékét (dátum és idő kombináció), illetve vizuálisan kiemeli a kiválasztott időpontot.

Itt kihívásként említeném a felhasználó számára megjelenített elérhető időpontok megjelenítését. Azt kellett megoldanom, hogy ne csúszhassanak egymásba az időpontok. Tehát például ha le van foglalva valamilyen kezelésre egy időpont az adott

dátumon 11:00-ra, akkor ne jeleníthessen meg a felhasználónak 10:45-re szabad időpontot, amikor egy 35 perc hosszúságú kezelést választott ki. Ezt az 5.4. ábrán látható módszerrel oldottam meg, amit az "idopontfoglalas.html" JS kódja hív meg. A módszer először is 15-tel osztható számra kerekíti a kiválasztott kezelés hosszát felfelé, (például egy 50 perces kezelés hosszát 60 percesre kerekíti), majd a HTML oldalon lévő JS kód által megjelenített időpontokat szűri. A `get_available_slots` módszer biztosítja, hogy csak azok az időpontok jelenjenek meg a felhasználónak, amelyek nem ütköznek más foglalásokkal. Ehhez a módszer minden egyes időintervallumot ellenőriz, hogy az adott időpontban és az azt követő időtartamban (a kezelés hossza alapján) ne legyen másik foglalás. A módszer figyelembe veszi az orvos munkaidejét is, így csak az orvos által megadott munkaidőn belüli időpontokat jeleníti meg. Az eredmény egy JSON válasz, amely tartalmazza az elérhető időpontokat és azok elérhetőségét. Ez a válasz a frontend JavaScript kódjában kerül feldolgozásra, amely a felhasználó számára vizuálisan is megjeleníti a szabad időpontokat.

5.4. ábra. A get_available_slots metódus implementációja a views.py-ban

```
101 def get_available_slots(request):
102     doctor_id = request.GET.get('doctor')
103     date = request.GET.get('date')
104     treatment_id = request.GET.get('treatment')
105     treatment = Treatment.objects.get(id=treatment_id)
106     treatment_duration = treatment.duration.total_seconds() // 60
107
108     # Felfelé kerekítés 15 perces intervallumokra
109     treatment_duration = ((treatment_duration + 14) // 15) * 15
110
111     working_hours = WorkingHours.objects.filter(doctor_id=doctor_id, date=date)
112     slots = []
113
114     for wh in working_hours:
115         start_time = datetime.combine(wh.date, wh.start)
116         end_time = datetime.combine(wh.date, wh.end)
117         current_time = start_time
118
119         while current_time + timedelta(minutes=treatment_duration) <= end_time:
120             slot_end_time = current_time + timedelta(minutes=treatment_duration)
121             is_available = not Appointment.objects.filter(
122                 practitioner_id=doctor_id,
123                 start__lt=slot_end_time,
124                 end__gt=current_time,
125                 status='booked'
126             ).exists()
127             slots.append({
128                 'time': current_time.strftime('%H:%M'),
129                 'available': is_available
130             })
131             current_time += timedelta(minutes=15)
132
133     return JsonResponse({'slots': slots})
```

5. Fizetési mód kiválasztása:

- Interakció: A fizetési módok közül a felhasználó egyet választ (pl. online fizetés – PayPal vagy bankkártya, illetve helyszíni fizetés).
- Kód: A JavaScript eseménykezelő beállítja a "payment_method" mezőt, amely később a Django backendben a foglalás véglegesítéséhez és a fizetési folyamat elindításához szükséges.

6. Foglalás véglegesítése:

- Interakció: A felhasználó a „Foglalás” gombra kattint, miután minden kötelező adatot megadott.
- Kód:

- A JavaScript egy megerősítő üzenetben összegzi a kiválasztott orvos, kezelés, dátum és időpont adatait.
- Ha a felhasználó megerősít, az űrlap elküldésre kerül, és a views.py "idopontfoglalas" függvénye végrehajtja az alábbiakat:
 - * Ellenőrzi az időpont elérhetőségét (szabad-e az adott időintervallum).
 - * Lekéri az orvos munkaidejét a WorkingHours modell alapján.
 - * Létrehoz egy új Appointment objektumot az adatbázisban.
 - * Rögzíti a foglalás fizetési státuszát a PaymentStatus modell segítségével.
 - * Ha a felhasználó online fizetést választott, átirányítja a fizetési oldalra a tranzakció elindításához.
- "updateAvailableSlots()" JavaScript függvény: Lekéri a kiválasztott orvos, kezelés és dátum alapján az elérhető időpontokat az API-ból, majd megjeleníti a délelőtti és délutáni időpontokat gombok formájában.
- "earliestAppointmentButton" eseménykezelő: A „Leghamarabbi időpont kiválasztása” gomb megnyomásakor az API-tól lekéri a legkorábbi szabad időpontot, majd automatikusan beállítja a dátumot és az időpontot.
- Űrlap beküldése: A foglalási űrlap beküldése előtt a JavaScript egy megerősítő párbeszédablakban összegzi a választott opciókat, majd a felhasználó jóváhagyása esetén elküldi az űrlapot. A Django view feldolgozza a POST kérést, létrehozza az időpontfoglalást, és ha a felhasználó az online fizetést választja, akkor a fizetési oldalra irányítja át, ha pedig a fizetés a helyszínen opciót, akkor a profil oldalra.

5.4. A fizetési oldal

A fizetési oldal a "payment_page.html" oldalon lett megvalósítva. A fizetéshez a tématervembe OTP Simple Pay fizetési rendszert írtam, viszont egy kis kutatás után a PayPal-t könnyebben beépíthetőbbnek, stabilabbnak, és jobban átláthatóbbnak találtam. Ezért inkább azt építettem be az alkalmazásba.

A PayPal fizetés demózásához létre kellett hoznom egy sandbox PayPal fiókot, majd az azonosítót használva importálnom kellett az oldalra egy JavaScriptet az alábbi módon:

Kódrészlet 5.1. A PayPal sandbox importja

```
<script src="https://www.paypal.com/sdk/js?client-id=Abw9kvI2SEa...&currency=HUF"></script>
```

Ezután egy "div"-hez hozzá kellett adnom a "paypal-button-container" id-t, és meg kellett valósítanom a fizetési logikát, amit az alábbi módon tettem meg a HTML fájlban:

Kódrészlet 5.2. A fizetési logika

```
<script>
paypal.Buttons({
  createOrder: function(data, actions) {
    return actions.order.create({
      purchase_units: [{
        amount: {
          value: '1', // Teszt összeg
          currency_code: 'HUF'
        }
      }]
    });
  },
  onApprove: function(data, actions) {
    return actions.order.capture().then(function(details) {
      fetch("{% url 'payment_callback' %}", {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          'X-CSRFToken': '{{ csrf_token }}'
        },
        body: JSON.stringify({
          orderID: data.orderID,
          status: details.status,
          orderRef: '{{ appointment.id }}'
        })
      }).then(response => {
        if (response.ok) {
          alert('A tranzakció sikeresen megtörtént: ' + details.payer.name + given_name);
          window.location.href = "{% url 'profile' %}";
        } else {
          alert('Hiba történt a fizetés során.');
```

```
        console.error('Hiba történt a fizetes soran:', error);
        alert('Hiba történt a fizetes soran.');
```

```
    });
  });
},
  onError: function(err) {
    console.error(err);
    alert('Hiba történt a fizetes soran.');
```

```
  }
}).render('#paypal-button-container');
```

```
</script>
```

Ennek hatására a fizetési oldalon egy bankkártya, vagy egy PayPal fiókos fizetési opció jelenik meg, amiből a felhasználó tud választani. Ha a fizetés sikeres, akkor az időponthoz tartozó "PaymentStatus"-nak az *is_paid* adattagját "true"-ra állítja, és a felhasználót átirányítja a Profil oldalra, ahol a lefoglalt időpontnál látszik a "Fizetve" felirat. Ha nem sikerült a fizetés akkor kiírja a hibaüzenetet, és nem állítja az *is_paid* adattagot "true"-ra.

5.5. A profil oldal

A profil oldal mindhárom felhasználói jogosultságú felhasználó számára látható, viszont mindhárom típusú felhasználó számára más jelenik meg rajta. A "ProfileForm" mindenképp megjelenik, ezen tudja változtatni a felhasználó a felhasználónevét, az email címét, és a jelszavát. Ez a form szerkeszti a "RendeloUser" modellt. A "profile_view" metódus a views.py fájlban az 5.5. ábrán látható módon vizsgálja meg a felhasználó jogosultságát, hogy ez alapján jelenítse meg a formokat a felhasználó számára. Ha a felhasználó "superuser", akkor semmit nem jelenít meg a "ProfileForm" után (a *patient_form* változó amúgy is none kezdőértékű), ha nem "superuser", hanem "staff", akkor ez azt jelenti, hogy egy orvos kattintott az oldalra, tehát lekéri azt a "Doctor" osztálypéldányt az adatbázisból, amelyiknek egyezik az azonosítója a bejelentkezett felhasználó azonosítójával, és a *doctor_form* változóba is lekéri az adott formot, és átadja neki a "Doctor" példányt, amit lekért. Ha pedig nem is "staff" a felhasználó, akkor a "Patient" példányt és formot kéri le hozzá.

5.5. ábra. A profile_view metódus form kiválasztása

```
401     if user.is_superuser:
402         patient_form = None
403     elif user.is_staff:
404         doctor = get_object_or_404(Doctor, id=user.id)
405         doctor_form = DoctorForm(request.POST or None, request.FILES or None, instance=doctor)
406     else:
407         try:
408             patient = get_object_or_404(Patient, id=user.id)
409             patient_form = PatientForm(request.POST or None, instance=patient)
410         except Patient.DoesNotExist:
411             patient_form = None
```

A formok kitöltve jelennek meg a felhasználók adataival az oldalon, amiket változtathatnak és a formok után található "Mentés" gombbal a változtatásaik elmenthetők.

"User" szintű felhasználó esetében megjelennek a formok után a felhasználó által lefoglalt időpontok is, dátum szerint csökkenő sorrendben. Az időpontok kattinthatók, ezek átvisznek az "edit_appointment.html" "read only" nézetébe, ahol a felhasználó megtekintheti az adott időponthoz írt kezelés leírást, és az időpont adatait bővebben. Például hogy fizetve van-e, vagy hogy melyik orvoshoz lett foglalva. Az időpontok le is mondhatók a "Lemondás" gombbal a profil oldalon abban az esetben, ha dátum szerint legalább egy nappal későbbre lettek lefoglalva. Az időpont lemondásához megerősítést kér az oldal egy párbeszédablakban. Az időpont törlését a views.py fájlban található "cancel_appointment" metódus végzi.

6. fejezet

Staff-szintű felhasználói felületek, és azok működése

Ahogy a korábbiakban már leírtam, az alkalmazásban a Staff felhasználói szint az orvosokat reprezentálja. Nekik lehetőségük van belenézni bármelyik páciens kezeléstörténetébe, és hozzáférnek a páciensek elérhetőségéhez is. Továbbá megadhatják a saját munkaidejüket, és megnézhetik a saját időpontjaikat dátumonként szűrve. Ugyanakkor időpontfoglaláshoz nincs joguk, nem is lenne lehetséges, mivel az orvosok fiókjaihoz nincs "Patient" példány rendelve.

6.1. A "Páciensek" oldal

Ezen az oldalon az orvos rákereshet bármelyik páciensre név szerint az oldalon lévő kereső segítségével, vagy az "Összes páciens" gombra kattintva visszatérhet az alapértelmezett nézetbe, ami az összes adatbázisban lévő páciens kilistázza az oldalon látható táblázatba. Az oldal a "patients.html" fájlban lett megvalósítva. Mindegyik páciens kattintható, kattintás hatására az adott páciens adatlapjára jut a felhasználó, ami a "patient.html" oldalon lett megvalósítva. Ezen az oldalon vannak a páciensnek az adatai láthatók amelyikre a felhasználó kattintott, továbbá az összes lefoglalt időpontja és azoknak leírása időrend szerint csökkenő sorrendben. Az oldalon alapértelmezetten minden a páciens által lefoglalt időpont megjelenik, viszont a felhasználónak lehetősége van hónap alapján szűrni őket, ezzel könnyebbé téve a kezeléstörténet vizsgálatát. Az oldalon található még egy

"Vissza a páciensekhez" gomb, ami visszavezet a "Páciensek" oldalra. A 6.1. ábrán ennek az oldalnak a kinézete látható.

6.1. ábra. A "Páciensek" oldal

Név	Nem	Születési dátum	Elérhetőség
Sándor Márton	Férfi	Dec. 26, 2003	06306766143
BFAM77	Férfi	March 28, 2004	06309877435

6.2. Az "Időpontjaim mára" oldal

Ezen az oldalon láthatja az orvos a hozzá foglalt időpontokat a megadott dátumon.

6.2. ábra. Az "Időpontjaim mára" oldal

Válassz egy dátumot:

2025. 04. 09.

Szűrés

Sándor Márton 2025-04-09 08:00 - 08:30

Kezelés: Tejfog trepanálás

Orvos: Dr. Kovács Alexandra

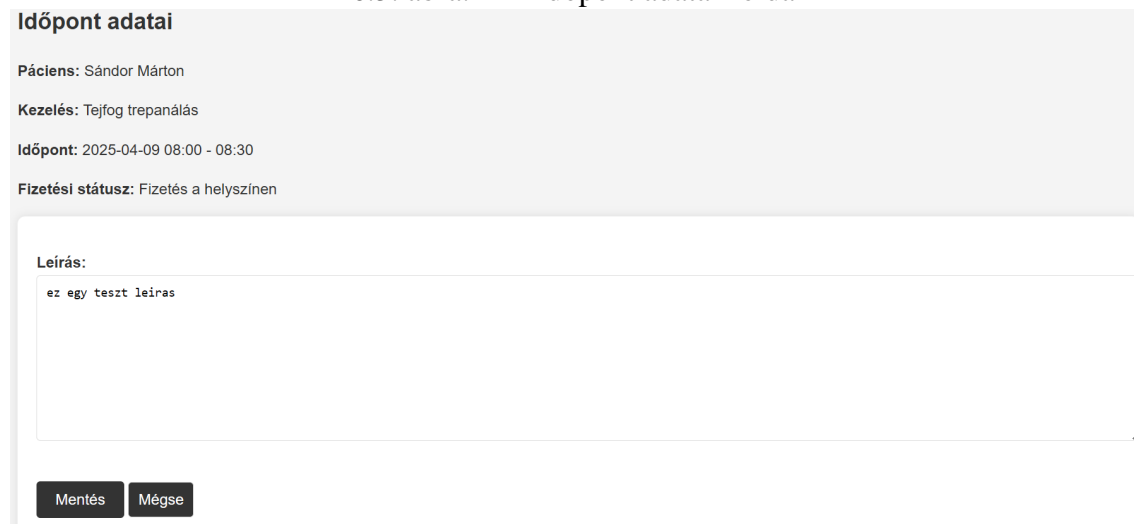
Leírás: ez egy teszt leírás

Szerkesztés

A 6.2. ábrán látható módon jelennek meg az időpontok az oldalon. Alapértelmezetten a jelenlegi dátumra lefoglalt időpontok jelennek meg, viszont a beviteli mező segítségével a más dátumokra foglalt időpontokat is meg lehet nézni. Egy adott időpontnál a "Szerkesztés" gomb átnavigálja a felhasználót az "Időpont adatai" oldalra, ahol a "user" típusú

felhasználók csak nézhetik az időpontjaik adatait, viszont az orvosoknak lehetőségük van szerkeszteni azok leírását. Így adható meg a páciens kezeléstörténete. Ez az oldal az "edit_appointment.html" oldalon lett megvalósítva.

6.3. ábra. Az "Időpont adatai" oldal



Időpont adatai

Páciens: Sándor Márton

Kezelés: Tejfog trepanálás

Időpont: 2025-04-09 08:00 - 08:30

Fizetési státusz: Fizetés a helyszínen

Leírás:

ez egy teszt leírás

Mentés Mégse

A "Mentés" gombbal elmenthető a leírás amit az időponthoz rendeltünk, a "Mégse" gomb pedig visszavigáz az "Időpontjaim mára" oldalra.

6.3. A "Munkaidő oldal"

Az orvos a munkaidejét ezen az oldalon állíthatja be. Az oldal a "working_hours.html" fájlban lett megvalósítva.

6.4. ábra. A munkaidő megadása

Amikor a felhasználó ellátogat az oldalra és kiválaszt egy dátumot, akkor a 6.4. ábrán látható kép fogadja. A rendelő 8:00-20:00 között van nyitva, tehát ennél korábbi kezdést, vagy későbbi végzést nem lehet beállítani munkaidőnek. A kezdés és végzés ideje a + és - gombokkal növelhető, vagy csökkenthető. Egy kattintás 15 perccel növeli, vagy csökkenti a beviteli mezőben látható időt. A beviteli mező ezeken a gombokon kívül mással nem szerkeszthető. Ez azért van, hogy a felhasználó ne adhasson meg magának olyan kezdési, vagy végzési időpontot, ami nem "kerek", és az időpontfoglalási rendszert összezavarná. A munkaidőt beállítani, vagy a már beállítottat szerkeszteni a "Mentés" gombbal lehet. Ennek a hatására az alkalmazás elmenti a "WorkingHours" modellbe az orvos munkaidejét. Abban az esetben, ha még a kiválasztott napra nincs megadva az adott orvosnak munkaidő, akkor megjelenik a képen látható üzenet is az oldalon.

6.4. Az orvosok Profil oldala

Az orvosok profil oldala ugyan abban a fájlban lett megvalósítva ugyan azzal a működési elvvel, mint az az oldal, amit az 5.5. fejezetben bemutatam. Azonban ennél a típusú felhasználónál akadtak nehézségek, mivel a "DoctorForm" form töltődik be a "PatientForm" helyett, és ennek a formnak kezelnie kell képeket. Itt is a Django beépített fájlbeviteli mezőjét szerettem volna használni, de azt nem sikerült CSS kóddal designolnom, és nem nyújtott kellően szép látványt. A megoldásom az lett erre a problémára,

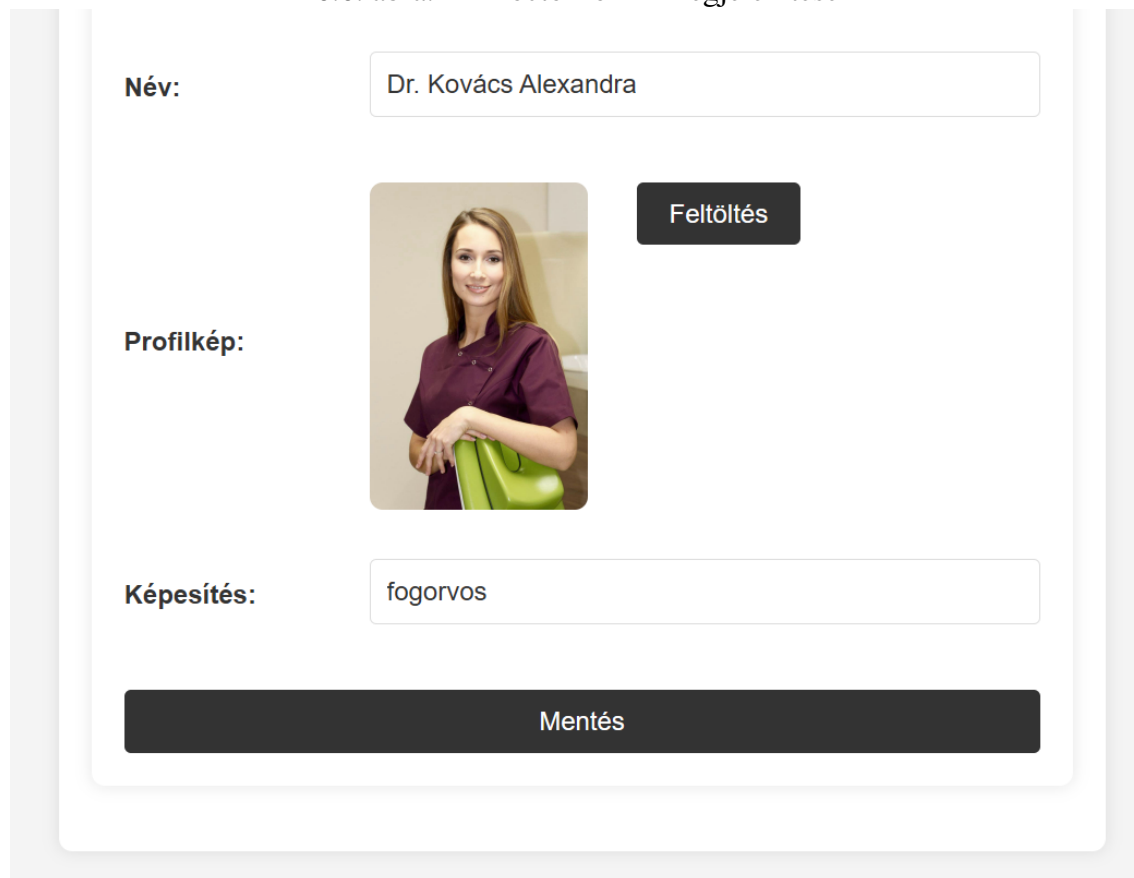
hogy készítettem egy saját képbeviteli oldalt a "custom_clearable_file_input.html" fájlba, készítettem a "forms.py"-ba egy osztályt hozzá, és a 6.5. ábrán látható módon beleraktam a "DoctorForm"-ba, hogy ezt használja a képek beviteléhez.

6.5. ábra. A "DoctorForm"

```
124 class CustomClearableFileInput(forms.ClearableFileInput):
125     template_name = 'custom_clearable_file_input.html'
126
127 class DoctorForm(forms.ModelForm):
128     class Meta:
129         model = Doctor
130         fields = ['name', 'photo', 'qualification']
131
132     name = forms.CharField(max_length=255, label='Név')
133     photo = forms.ImageField(label='Fénykép', required=False, widget=CustomClearableFileInput)
134     qualification = forms.CharField(max_length=255, label='Képesítés')
135
136     def __init__(self, *args, **kwargs):
137         super().__init__(*args, **kwargs)
138         self.fields['photo'].widget.attrs.update({'class': 'custom-file-input'})
139         self.fields['photo'].label = 'Fénykép'
140         self.fields['photo'].help_text = ''
```

Ezt már lehetett CSS kóddal designolni, így megoldódott a probléma, és a 6.6. ábrán látható is a munka eredménye.

6.6. ábra. A "DoctorForm" megjelenítése



The screenshot displays the 'DoctorForm' interface within a light gray frame. It features three main input sections: 1. 'Név:' (Name) with a text box containing 'Dr. Kovács Alexandra'. 2. 'Profilkép:' (Profile picture) with a placeholder image of a woman in a purple medical uniform and a dark gray 'Feltöltés' (Upload) button to its right. 3. 'Képesítés:' (Qualification) with a text box containing 'fogorvos' (dentist). At the bottom, a large dark gray button labeled 'Mentés' (Save) is centered.

A "Feltöltés" gomb megnyomásával ki lehet választani a képet, amit a felhasználó profilképként szeretne használni. Ez a kép jelenik meg a pácienseknek az időpont foglalásnál is, amikor az orvos kiválasztására kerül sor. A program a profilképeket a `rendelo/media/-doctor_pictures` mappába menti.

7. fejezet

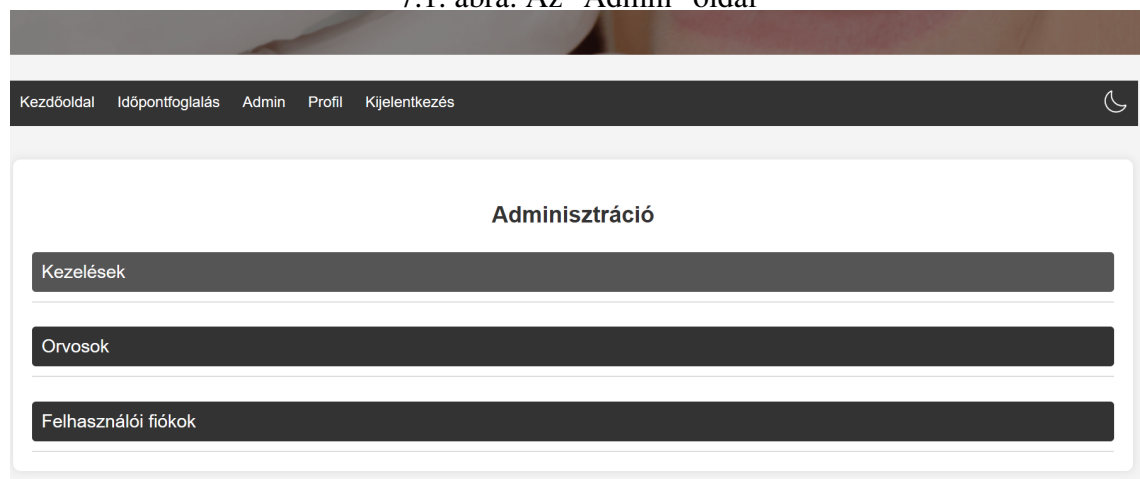
Superuser-szintű felhasználói felületek, és azok működése

A Superuser szintű felhasználók az oldal adminisztrátorai. Nem tartozik a profiljukhoz sem "Patient" példány, sem "Doctor" példány. A felhasználók személyes adatain kívül mindenhez hozzáférnek az adatbázisban, és mindent joguk van törölni, vagy módosítani. A Django beépített admin felületére is van joguk bejelentkezni. Időpontot foglalni viszont nem tudnak, hasonló okok miatt, mint az orvosok.

7.1. Az "Admin" oldal

Az Admin oldal az "admin.html" fájlban lett megvalósítva, az adminisztrátor ezen az oldalon éri el az összes admin felületet.

7.1. ábra. Az "Admin" oldal



Az oldalon a 7.1. ábrán látható három lenyíló menü fogadja a felhasználót. Ezekben vannak a kezelések, az orvosok, és a felhasználói fiókok szerkesztési lehetőségei.

Ha lenyitja a kezeléseket akkor kilistázza az oldal az adatbázisban található összes megadott kezelést. A lenyíló gomb alatt található egy "Új kezelés hozzáadása" gomb, ami az "add_treatment.html" oldalra navigálja a felhasználót. Ezen az oldalon a "Treatment-Form" található, amiben megadhatja a hozzáadni kívánt kezelés adatait, majd a "Mentés" gombbal elmentheti, vagy a "Mégsem" gombbal visszatérhet az Admin oldalra.

A kilistázott kezeléseknél mindegyikhez jut két gomb: "Törlés", vagy "Szerkesztés".

A "Törlés" a "Delete_treatment.html" fájlba navigálja a felhasználót, ahol megkérdezi az oldal, hogy biztosan törölni kívánja-e a kezelést, és megjeleníti a kezelés adatait is. Az oldalon található "Törlés" gomb értelem szerűen törli, míg a "Mégsem" gomb visszavigál az Admin oldalra.

A "Szerkesztés" gomb az "edit_treatment.html" oldalra navigál, ahol szintén megjelenik a "TreatmentForm" a kiválasztott kezelés adataival kitöltve. Bármilyen adatot átírhat itt is a felhasználó, az oldalon lévő "Mentés" és "Mégsem" gombok pedig a megszokott módon működnek.

Az orvosok lenyíló menüben is hasonló lehetőségek találhatók, kilistázza az oldal az összes adatbázisban szereplő orvost egymás alá, és lehet újat elmenteni, szerkeszteni, és törölni az adatbázisból. Az új hozzáadása és a szerkesztés a "ProfileForm", és "Doctor-Form" segítségével történik. Ezekhez a műveletekhez is külön oldalakat készítettem az alábbi fájlokban:

- "add_doctor.html"
- "edit_doctor.html"
- "delete_doctor.html"

Fontos, hogy itt az új orvos hozzáadása nem csak egy új "Doctor" példányt ad hozzá az adatbázishoz, hanem először létrehoz egy "RendeloUser"-t is, és annak az *id*-jával hozza létre a "Doctor"-t, hogy az orvos azonnal használhassa a felhasználói fiókját is. Az új orvos létrehozása esetén az orvos a megadott email címére meg is kap egy emailt, amiben megkapja a belépési adatait, hogy meg tudja őket változtatni.

A felhasználói fiókok lenyíló menüben ismét az adatbázisban található összes adat fogadja a felhasználót kilistázva, viszont ezeket lehetősége van szűrni a megszokott módon felhasználónév alapján.

7.2. ábra. Felhasználói fiókok

The screenshot shows the Django Admin interface for user accounts. At the top, there's a dark header with the text 'Felhasználói fiókok'. Below it, there's a button 'Új admin felhasználó hozzáadása'. A search bar is present with the placeholder text 'Keresés felhasználónév alapján'. Below the search bar are two buttons: 'Keresés' and 'Összes felhasználó'. The main content area displays a list of users. Each user entry includes a username, email, and role, along with a link to edit the profile. The users listed are:

username	email	role	actions
sandormarton265@gmail.com	sandormarton265@gmail.com	Superuser	Saját profil, szerkeszthető a "Profil" oldalon.
doki1	dokiemail@gmail.com	Orvos	Orvos, szerkeszthető az "Orvosok" menüben!
felhasznalo1	h269823@stud.u-szeged.hu		Szerkesztés Törlés

Az adminisztrátornak joga van új admin felhasználót hozzáadnia az adatbázis-hoz, amit az "Új admin felhasználó hozzáadása" gombbal tehet meg. Ez elvezeti az "add_admin_user.html" oldalra, amin a "CustomUserCreationForm" segítségével a megszokott módon létrehozhat új rekordot az adatbázisba.

A kilistázott felhasználók közül itt nem szerkeszthetők az orvosok, mivel azokhoz ott van az orvosok szerkesztési oldala, és itt nem szerkeszthető, vagy törölhető az admin saját fiókja sem. Ez egy védelem, hogy az alkalmazás ne maradjon adminisztrátor nélkül.

A szerkesztést viszont meg tudja oldani a saját "Profil" oldalán, ami ugyan úgy működik, mint a többi szintű felhasználó esetében.

A "user" szintű felhasználókat, és a többi adminisztrátort pedig ugyan úgy tudja szerkeszteni a "ProfileForm", és a "PatientForm" segítségével, vagy törölni a megszokott módon. Ezekhez is készítettem két külön oldalt:

- "edit_user.html"
- "delete_user.html"

8. fejezet

Design

A Design-t összesen 3 CSS fájlal oldottam meg. Ezek a fájlok a `rendelo/static/Css` mappában találhatók:

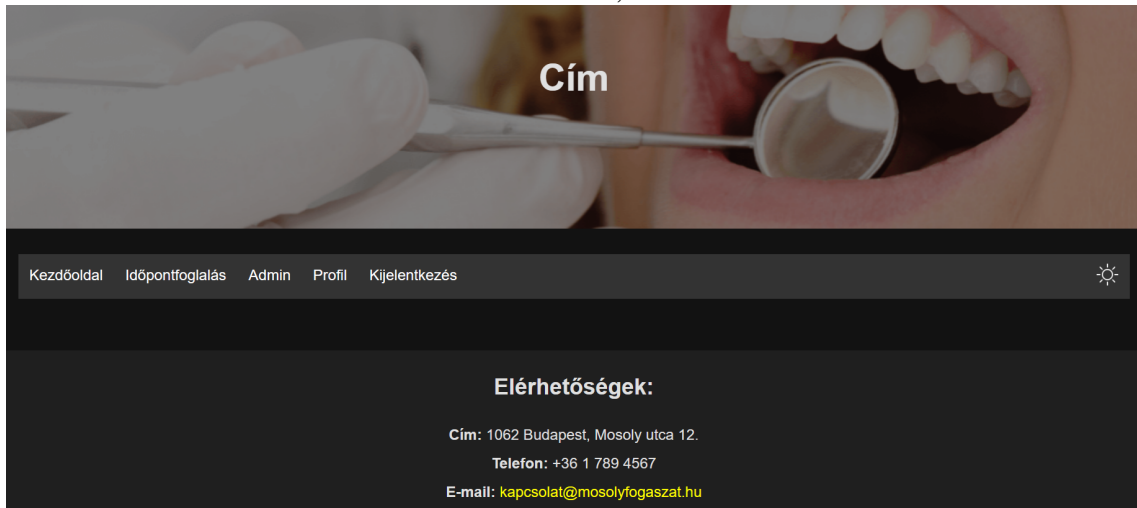
- `style.css`: Az oldalak kinézetét írtam meg benne színek nélkül.
- `colors_light.css`: Az alkalmazás világos módjának a színbeállításait írtam meg benne.
- `colors_dark.css`: Az alkalmazás világos módjának a színbeállításait írtam meg benne.

8.1. ábra. A "base.html" kinézete, és a navbar világos módban



A 8.1. ábrán világos módban megtekinthető a "base.html". Ahogy korábban már írtam, ebből származik az összes többi oldal, így mindegyik ezt a designt követi. Sötét módba a navigációs sáv jobb oldalán lévő "hold" ikonnal léphetünk.

8.2. ábra. A "base.html" kinézete, és a navbar sötét módban



Ezt a "base.html" fájlban megírt JS kód hajtja végre:

Kódrészlet 8.1. A színösszeállítások között váltó JS kód

```
<script>
function toggleColorScheme() {
    const currentScheme = document.getElementById('color-scheme').getAttribute('href')
    ;
    const newScheme = currentScheme.includes('colors_light.css') ? 'colors_dark.css' :
        'colors_light.css';
    document.getElementById('color-scheme').setAttribute('href', `{{ static 'css/' }}`
        + newScheme);
    localStorage.setItem('color-scheme', newScheme);
    updateIcon(newScheme);
}

function updateIcon(scheme) {
    const icon = document.getElementById('color-scheme-icon');
    if (scheme.includes('colors_light.css')) {
        icon.src = `{{ static "images/moon_icon.png" }}`;
        icon.alt = 'Sötét mód';
    } else {
        icon.src = `{{ static "images/sun_icon.png" }}`;
        icon.alt = 'Világos mód';
    }
}
}
```

```
document.addEventListener('DOMContentLoaded', function() {  
  const savedScheme = localStorage.getItem('color-scheme') || 'colors_light.css';  
  document.getElementById('color-scheme').setAttribute('href', `${% static 'css/' %}`↵  
    + savedScheme);  
  updateIcon(savedScheme);  
});  
</script>
```

Alapértelmezetten a világos mód Css fájlját importálja, viszont ha kattintunk a "hold" ikonra, akkor átírja az importot a sötét mód Css fájljának az importjára, majd utána átváltja "napra" az ikont, amire kattintottunk jelezve ezzel, hogy az a világos módba vezet. A design megírásához sima Css-t használtam.

9. fejezet

Összefoglaló

A szakdolgozatom elején bemutatam a Django keretrendszer által nyújtott lehetőségeket, az MVT architektúra működését, valamint a keretrendszer által biztosított eszközöket, amelyek megkönnyítik a webalkalmazások fejlesztését. Ismertettem a Django ORM működését, amely lehetővé teszi az adatbázis-kezelést SQL ismeretek nélkül, valamint bemutatam az alkalmazás adatbázisának FHIR szabvány szerinti felépítését.

Ezt követően részletesen bemutatam az alkalmazás felhasználói felületeit, amelyek a páciensek, orvosok és adminisztrátorok számára készültek. Ismertettem az időpontfoglalási rendszer működését, amely lehetővé teszi a páciensek számára, hogy egyszerűen foglaljanak időpontot a rendelőben elérhető kezelésekre. Bemutattam az online fizetési rendszer integrációját, amely a PayPal API segítségével biztosít biztonságos és egyszerű fizetési lehetőséget.

A szakdolgozatban külön fejezetet szenteltem az adminisztrátori felületek ismertetésére, amelyek lehetővé teszik a kezelések, orvosok és felhasználók adatainak kezelését. Részletesen bemutatam az orvosok munkaidő-beállítási lehetőségeit, valamint a páciensek kezeléstörténetének megtekintésére és szerkesztésére szolgáló funkciókat.

A dolgozatban kitértem az alkalmazás reszponzív designjára, amely modern CSS és JavaScript megoldásokkal készült, biztosítva a könnyű használatot különböző eszközökön. Bemutattam a világos és sötét mód közötti váltás megvalósítását, amely a felhasználói élményt hivatott javítani.

A szakdolgozat elkészítése során betekintést nyertem a Django keretrendszerben való fejlesztés rejtelseibe, valamint a webfejlesztés különböző aspektusaiba. Megismertem

az online fizetési rendszerek integrációjának folyamatát, az adatbázis-tervezés kihívásait, valamint a felhasználói jogosultságok kezelésének megvalósítását. A dolgozatom forráskódja elérhető a GitHubon, ahol részletes dokumentáció is található a projekt működéséről és telepítéséről.

Nyilatkozat

Alulírott Sándor Márton programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Szeged, 2025. május 19.

.....
aláírás

Irodalomjegyzék

[1] Django: The web framework for perfectionists with deadlines . <https://www.djangoproject.com>.

[2] FHIR-Fast Healthcare Interoperability Resources. <https://hl7.org/fhir/index.html>.

[3] How Django's MVT Architecture Works: A Deep Dive into Models, Views, and Templates. <https://www.freecodecamp.org/news/how-django-mvt-architecture-works/>.

[4] Python Documentation. <https://docs.python.org/3/>.

[5] Using SQLite as a Database Backend in Django Projects. <https://medium.com/@codewithbushra/using-sqlite-as-a-database-backend-in-django-projects-code-with-bu>