Gymnázium, Praha 6, Arabská 14

Programování

TÝMOVÝ ROČNÍKOVÝ PROJEKT

2024

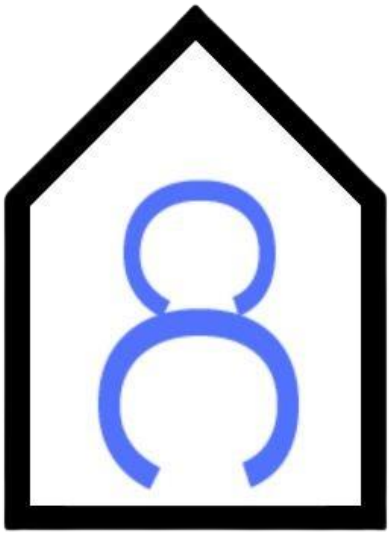
Oliver Hurt, Sabina Javůrková, David Mikolášek

Gymnázium, Praha 6, Arabská 14

Arabská 14, Praha 6, 160 00

TEAMOVÝ ROČNÍKOVÝ PROJEKT

CARE COMPANION



Autoři: Oliver Hurt, Sabina Javůrková, David Mikolášek

Třída: 3.E

Předmět: Programování

Školní rok: 2023/2024

Vyučující: Mgr. Jan Lána

Prohlašujeme, že jsme ročníkovou práci vypracovali samostatně pod vedením Mgr. Jana Lány a všechny použité zdroje jsou poctivě ocitovány. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů udělujeme bezúplatně škole Gymnázium, Praha 6, Arabská14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne .....

.....

.....

Podpisy autorů

Anotace

Cílem ročníkové práce bylo naprogramovat webovou aplikaci, která bude nápomocná v domově pro seniory, a to pro pracovníky i pro klienty, a bude plně responsivní na mobilních zařízeních, za použití programovacích jazyků Python, HTML a CSS. Na zhotovení práce byl použit framework Django, Bootstrap a databáze SQLite.

Abstract

The goal of the project was to make a web application that will be helpful in nursing homes for the workers and clients alike and will be fully responsive on mobile devices using Python, HTML and CSS. Django, Bootstrap and SQLite were used to make this project.

Zadání ročníkové práce

Webová aplikace, která bude nápomocná pro domov pro seniory. Přihlášení bude možné skrz učet opatrovníka nebo pacienta (senior). Opatrovník má možnost prohlížet lékařské karty pacientů, mapu pokojů, rozvrh směny a další užitečné věci, které mu práci ulehčí. Pacient uvidí seznam opatrovníků na směně a může si také zobrazit důležité kontakty na své blízké, svůj denní rozvrh a další užitečné věci.

Obsah

1. Úvod

Hlavním cílem naší ročníkové práce bylo zjednodušit práci opatrovníkům v domovech pro seniory a vytvořit prostředí, kde si klienti mohou přehledně procházet důležité informace. Mezi opatrovnické funkce naší aplikace patří přehled relevantních směn, zobrazení a možnost úpravy zdravotnických karet klientů, zobrazení a přidávání denních rozvrhů klientovi a jednoduchá příručka s užitečnými informacemi. Uživatel přihlášený jako klient má možnost si zobrazit kdo má zrovna směnu, v případě že opatrovník nemá zrovna směnu se zobrazí směna nejdřívější. Díky administrátorské části webu je snadné skrz admina vytvářet/mazat/upravovat uživatelské účty, směny, rozvrhy.

Tato dokumentace je rozdělena do dvou hlavních částí: backend a frontend. V kapitole o backendu jsou informace o nástrojích, které jsme k vytváření aplikace využili, přiblížení databáze a podrobné vysvětlení struktury projektu. Část rozebírající frontend zahrnuje informace o použitých technologiích, a hlavně vzhled aplikace.

2. Backend

Backend je část projektu, která řeší manipulaci s daty na straně serveru, logiku celého projektu, zpracování uživatelských požadavků a jednoduše všeobecnou funkcionalitu jakékoliv aplikace. Je to jádro celého projektu a zaslouží se za veškeré jeho chování. V našem projektu jsme využívali backend framework Django.

2.1 Využité nástroje

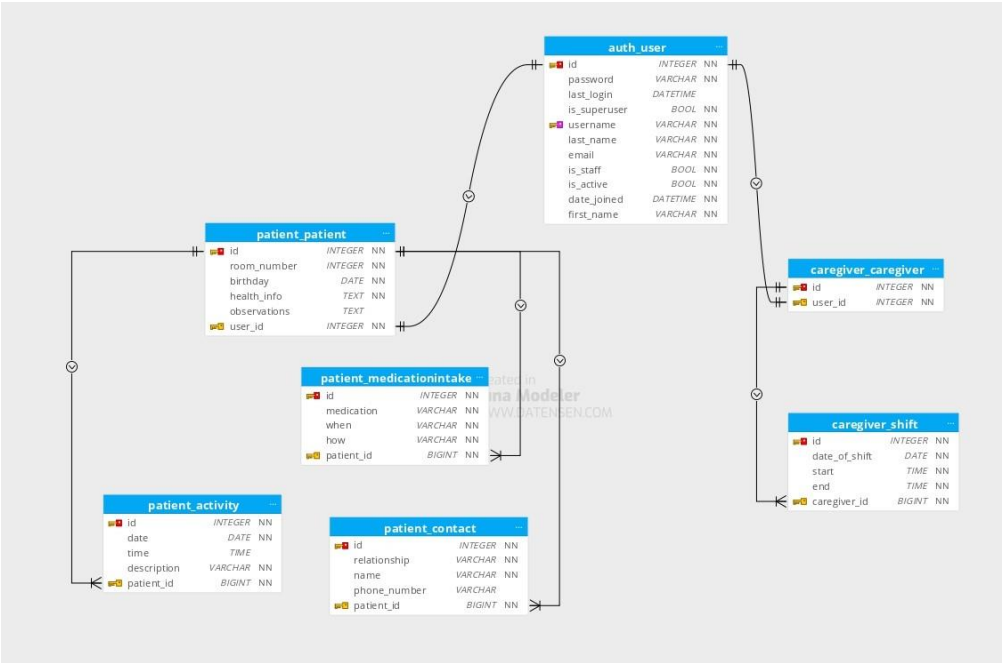
Django je open-source webový framework napsaný v jazyce Python, který usnadňuje vytváření komplexních webových aplikací. Poskytuje obsáhlou sadu nástrojů, čímž velmi usnadňuje vývojářům práci a zlepšuje jejich produktivitu. Django využívá konceptu "batteries-included", což znamená, že obsahuje mnoho předem napsaných modulů a knihoven pro běžné úlohy, jako je manipulace s databází, správa uživatelů, tvorba formulářů a zpracování HTTP požadavků. Jeho architektura je postavena na principu Model-View-Template (MVT), což pomáhá oddělit datovou logiku (models.py), vzhled (templates) a logiku řízení aplikace (views). [1]

Pro správný chod aplikace je nutno před samotným spuštěním nainstalovat patřičné knihovny. Pro jednoduchou instalaci náš repozitář obsahuje textový soubor requirements.txt, ve kterém jsou napsané názvy nepostradatelných balíčků. V našem případě je k funkčnosti aplikace potřeba mít nainstalované pouze dva balíčky, a to Django (framework) a Babel (formátování datumů do češtiny).

2.2 Databáze

Jako databázi jsme v našem projektu využívali SQLite. Je to jednoduchý databázový systém, který se často používá v kombinaci s frameworkem Django pro vytváření webových aplikací. Na rozdíl od tradičních databází, SQLite nepotřebuje samostatný serverový proces a ukládá se do jediného souboru na disku. To znamená, že je snadno přenosný a není potřeba složitého nastavení. V prostředí Django je SQLite často využíván pro vývoj a testování aplikací na lokálním počítači. Má jednoduchou syntaxi a je ideální pro menší projekty nebo ty, které nepotřebují složité databázové operace. [2]

Naše databáze se skládá z několika modelů navzájem na sobě závislých. Nejvýše postavený je Django model auth_user (uživatel). K němu se následně vztahují 2 modely – caregiver (opatrovník) a patient (klient). Tyto dva modely představují základ struktury naší databáze. Její schéma je možné vidět na Obrázku č. 1.



Obrázek č. 1 – Schéma databáze (vygenerované softwarem Luna Modeler)

Model patient představující klienta má atributy room_number (číslo pokoje), birthday (narozeniny), health_info (zdravotní informace), observations (postřehy o klientovi) a nakonec user_id (uživatelský účet) se kterým je ve vztahu přes foreign key. Modely activity (rozvrh) a contact (kontakt) jsou s klientem také spojeny přes foreign key, ale narozdíl od již zmíněných atributů mají další svoje atributy. Pod model rozvrhu (activity) spadají atributy date (datum), time (čas), description (popis) a patient_id (klient). Model kontaktu (contact) má atributy relationship (vztah), name (jméno), phone_number (telefonní číslo) a patient_id (klient).

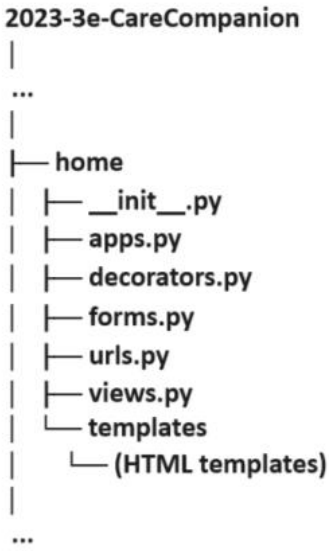
Druhý model caregiver, tedy opatrovník, nemá žádné vlastní atributy, jenom je přes foreign key spojen s modelem shift (směna), který má atributy date_of_shift (datum směny), start (začátek), end (konec) a caregiver_id (opatrovník).

2.3 Struktura projektu

Projekt je rozdělený na 3 takzvané “django applications” (nadále pouze aplikace): home, patient a caregiver. Každá z těchto aplikací reprezentuje část webu a pod její složku spadá několik souborů a podsložek. Tyto soubory většinou bývají __init__.py (její prezence znamená, že složka je balíček – je možné z ní něco importovat), models.py (tvorba modelů), views.py (řízení aplikace), apps.py (konfigurace nastavení pro aplikaci), urls.py (webové adresy) a admin.py (registrace a zobrazení modelů na django admin stránce). Časté podložky jsou templates (html soubory, zobrazení na frontendu) a migrations (migrace, modifikují databázi při změně v models.py).

2.3.1 Home

Aplikace home zahrnuje z větší části administrátorskou část webu, domovskou stránku aplikace Care Companion a přihlašování. V této složce jako jediné nenajdeme models.py a admin.py, protože pro administrátorský účet není potřeba speciálního ukládání informací. Grafické zobrazení hierarchie složky home je možné vidět na Obrázku č. 2



Obrázek č. 2 – Hierarchie složky aplikace home

V souboru urls.py jsou definovány webové adresy a následně je k nim přiřazena funkce z views.py. Pole urlpatterns inicializováno v tomto tedy obsahuje všechny web adresy, o jejíž chování se tato aplikace stará. Jsou zde definovány adresy jako domovská stránka naší aplikace a přihlášení, ale také veškeré adresy pro administrátorské účty. Některé z nich obsahují takzvané path converters (nadále pouze převodníky), což jsou části adresy, se kterýma se dá nějakým způsobem manipulovat (využití například při zobrazovaní informací ke konkrétnímu objektu) - viz Obrázek č. 3 (proměnná info_on_user je převodník typu String).

```
# Změna hesla konkrétního uživatele
path('sprava/uzivatele/<str:info_on_user>/zmena-hesla/', views.user_reset_password, name='user_reset_password'),
```

Obrázek č. 3 – převodník v urls.py (home)

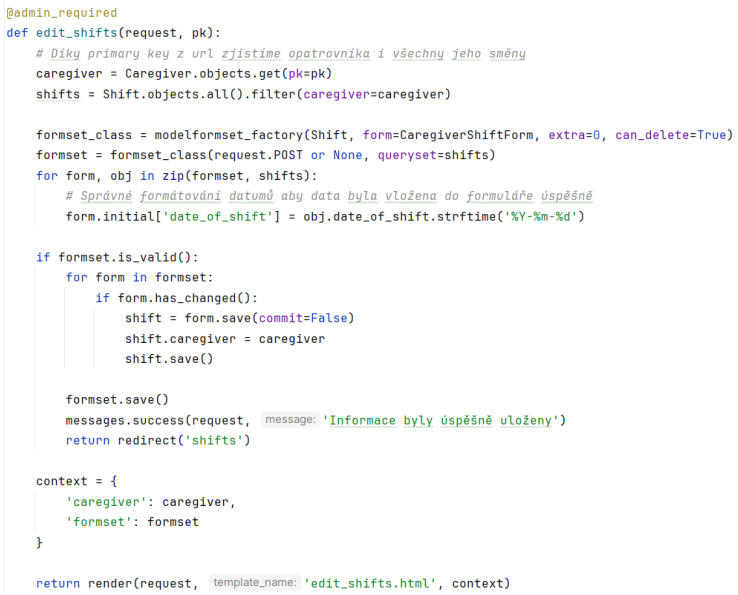
Soubor starající se o to, jak se bude projekt chovat a co bude posíláno do šablon ve složce templates je views.py. V aplikaci home je tento soubor velmi rozsáhlý a obsahuje především načítání objektů z databáze (zobrazení klientů, opatrovníků a adminů) a práce s formulářemi (tvorba účtů, přidání/úprava/mazání existujících účtů a informací o uživatelích). Výjimkou je funkce index (viz Obrázek č. 4), která se stará o to, aby se domovská stránka aplikace zobrazila pouze nepřihlášenému uživateli, v případě, že je přihlášen ho aplikace přesměruje na jeho vlastní domovskou stránku. Při přihlašování je zavolána funkce login_user, která zas přesměruje nově přihlášeného na jeho homepage podle toho, jaký je typ uživatele (viz Obrázek č. 5). A jako příklad funkce, která manipuluje s formulářem je fukce edit_shift, ta se vztahuje k objektu Shift. Načte si z URL převodník pk (primary key) a pode něj získá konkrétního opatrovníka. Pak jeho směnami naplní formulář a kontroluje, zda se “něco” ve formset nezměnilo (přidání čí smazání směny, k jednoduché úpravě se toto nevztahuje, jelikož se vyloženě nezmění formset) - viz Obrázek č. 6.

```python
# Homepage aplikace
# Pokud uživatel není přihlášený, tak se mu zobrazí homepage aplikace, pokud je, je přesměrován na svůj Home
1 usage  sabi
def index(request):
    if request.user.is_authenticated:
        if request.user.groups.filter(name='Admins').exists():
            return redirect('administration')
        elif request.user.groups.filter(name='Caregivers').exists():
            return redirect('index_caregiver')
        elif request.user.groups.filter(name='Patients').exists():
            return redirect('index_patient')
    else:
        return render(request, template_name: 'homepage.html')
```

Obrázek č. 4 – Funkce index ve views.py (home)

```python
# Přihlášení uživatele
# Jestli je přihlášení úspěšné, uživatel je podle jeho skupiny přesměrován na svoje Home url
1 usage  sabi +1
def login_user(request):
    if request.method == 'POST':
        username = request.POST.get("username").lower()
        password = request.POST.get("password")
        user = authenticate(request, username=username, password=password)

        if user is not None:
            login(request, user)

            if user.groups.filter(name='Admins').exists():
                return redirect('administration')
            elif user.groups.filter(name='Caregivers').exists():
                return redirect('index_caregiver')
            elif user.groups.filter(name='Patients').exists():
                return redirect('index_patient')
            # V případě, že uživatel existuje, ale nemá ani jednu ze skupin
            else:
                messages.success(request, message: 'Obraťte se na IT podporu.')
                return redirect('login_user')
```

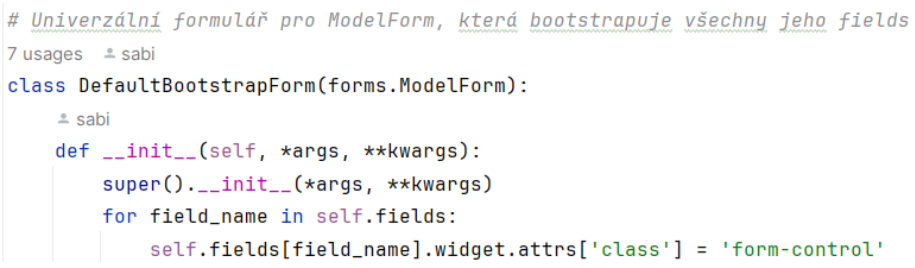
Obrázek č. 5 – Funkce login_user ve views.py (home)



```python
@admin_required
def edit_shifts(request, pk):
    # Díky primary key z url zjistíme opatrovníka i všechny jeho směny
    caregiver = Caregiver.objects.get(pk=pk)
    shifts = Shift.objects.all().filter(caregiver=caregiver)

    formset_class = modelformset_factory(Shift, form=CaregiverShiftForm, extra=0, can_delete=True)
    formset = formset_class(request.POST or None, queryset=shifts)
    for form, obj in zip(formset, shifts):
        # Správné formátování datumů aby data byla vložena do formuláře úspěšně
        form.initial['date_of_shift'] = obj.date_of_shift.strftime('%Y-%m-%d')

    if formset.is_valid():
        for form in formset:
            if form.has_changed():
                shift = form.save(commit=False)
                shift.caregiver = caregiver
                shift.save()

        formset.save()
        messages.success(request, message: 'Informace byly úspěšně uloženy')
        return redirect('shifts')

    context = {
        'caregiver': caregiver,
        'formset': formset
    }

    return render(request, template_name: 'edit_shifts.html', context)
```

Obrázek č. 6 – Funkce edit_shifts ve views.py (home)

Třídy formulářů týkající se tvorby/úpravy účtů, směn, aktivit, hesel a dalších věcí se všechny definují v souboru forms.py. Celý projekt obsahuje pouze jeden takovýto soubor, tím pádem v případě, že je nějaký formulář potřebný v jiné aplikaci (v opatrovnické části webu se využíva ObservationForm), importuje se odsud. Inicializace tříd formulářů je zpravidla velmi podobná u všech, pouze takzvaná fields, neboli kolonky jsou jiné. Jelikož je potřeba u každé kolonky každého formuláře nastavit bootstrap aby vše bylo na frontendu sympatičtější, je jako první definována třída DefaultBootstrapForm, která se efektivně stará o to, aby každá kolonka měla svůj boostrap – viz Obrázek č. 7. Na Obrázku č. 8 je vidět inicializace třídy pro formulář klientských aktivit PatientActivityForm, která právě třídu DefaultBootstrapForm dědí.



```python
# Univerzální formulář pro ModelForm, která bootstrapuje všechny jeho fields
7 usages  sabi
class DefaultBootstrapForm(forms.ModelForm):
    sabi
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        for field_name in self.fields:
            self.fields[field_name].widget.attrs['class'] = 'form-control'
```

Obrázek č. 7 – Třída DefaultBootstrapForm ve forms.py (home)

```python
# Formulář pro klientské rozvrhy
4 usages  sabi
class PatientActivityForm(DefaultBootstrapForm):
    sabi
    class Meta:
        model = Activity
        fields = ['date', 'time', 'description']
        labels = {
            'date': 'Datum',
            'time': 'Čas',
            'description': 'Popis'
        }
        widgets = {
            'date': forms.DateInput(attrs={'type': 'date'}),
            'time': forms.TimeInput(attrs={'type': 'time'})
        }
```

Obrázek č. 8 – Třída PatientActivityForm ve forms.py (home)

Soubor decorators.py v aplikaci home je také jediným takovým a obsahuje funkce, které jsou následně importovány do views.py všech aplikací projektu. Funkce group_required (viz Obrázek č. 9) je naprogramovaná tak, aby si vzala argument group_name (jméno skupiny = typ uživatele) a zkontrolovala, zda je uživatel součástí skupiny s tímto jménem. Pokud není, je uživatel přesměrován na stránku s přihlášením.

Následně jsou definovány 3 funkce admin_required, caregiver_required a patient_required a argumenty jsou jména skupin jednotlivých typů účtů.

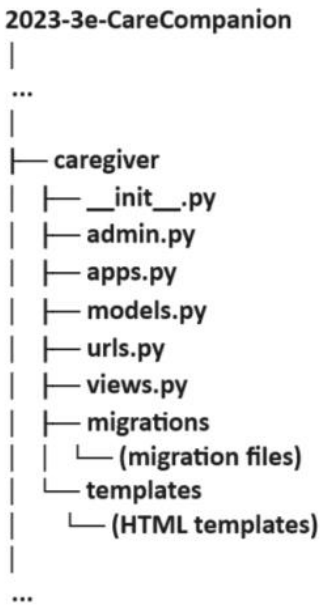
```python
# Univerzální funkce, která uživatelovi bez konkrétní skupiny zakáže přístup na určitou část webu
3 usages  sabi
def group_required(group_name):
    sabi
    def decorator(view_func):
        sabi
        def _wrapped_view(request, *args, **kwargs):
            if request.user.is_authenticated:
                groups = request.user.groups.values_list('name', flat=True)
                if group_name in groups:
                    return view_func(request, *args, **kwargs)

            return redirect('login_user')

        return _wrapped_view

    return decorator


# Inicializace funkcí, které kontrolují skupiny pro administrátory, opatrovníky a klienty
# Jsou využity ve views.py u většiny funkcí, jejíž přístup by měl být omezen
admin_required = group_required('Admins')
caregiver_required = group_required('Caregivers')
patient_required = group_required('Patients')
```

Obrázek č. 9 – Obsah decorators.py (home)

Složka templates obsahuje takzvané šablony, což jsou html soubory, které se renderují pomocí funkcí ve views.py. Každý šablonový soubor zpravidla patří k jedné webové adrese kromě universal_patient_form.html, který je využíván dvakrát, a to při tvorbě a úpravě klientských informací.

2.3.2 Caregiver

Druhá aplikace našeho projektu se nazývá caregiver a představuje opatrovnickou část webu. Mezi funkce opatrovníka patří zobrazení svých časově relevantních směn = současné či budoucí směny (dnešní směny jsou od ostatních graficky odlišné). Mají možnost si zobrazit zdravotnické karty všech registrovaných klientů a v nich upravovat takzvané "postřehy", které představují náhodná neodborná fakta o klientovi, jejíž znalost by se mohla ostatním opatrovníkům hodit. Další opatrovnická funkce je zobrazení a případná úprava nebo přidání klientských aktivit na konkrétní čas a datum. Nakonec je zde příručka, která obsahuje užitečné informace pro pečovatele. Grafické zobrazení hierarchie složky caregiver je možné vidět na Obrázku č. 10



Obrázek č. 10 – Hierarchie složky aplikace caregiver

Soubor urls.py obsahuje inicializaci webových adres spadajících pod opatrovnickou část webu. Stejně jako v aplikaci home se zde setkáme s převodníky, a to konkrétně u adres pro zobrazení jedné zdravotnické karty (viz Obrázek č. 11) a úpravu klientských rozvrhů. Pole urlpatterns obsahuje všechny webové adresy k výše zmíněným opatrovnickým funkcím.

```
# Jedna konkrétní karta klienta
path('karty-klientu/<str:full_name_of_patient>/', views.patient_info, name='patient_info'),
```

Obrázek č. 11 – Převodník v urls.py (caregiver)

O vytvoření takzvaného modulu Caregiver, který představuje opatrovníka se stará soubor models.py. V něm je obsažený kód, který definuje model Caregiver a Shift s jejich atributy (ty jsou vypsané a vysvětlené v kapitole Databáze). V modelové třídě Caregiver se navíc definují takzvané „properties" (česky vlastnosti) first_name a last_name, ty byly převzaté od asociovaného uživatelského účtu. Ve třídě pro model Shift je definovaná funkce, která vrací True/False podle toho, zda je konkretní směna noční (přes půlnoc) - viz Obrázek č. 12

```python
# Shift model
7 usages   sabi
class Shift(models.Model):
    date_of_shift = models.DateField()
    start = models.TimeField()
    end = models.TimeField()

    # Boolean funkce vracící jestli je směna přes noc - tato fce je využita ve views.py u opat. i klienta
    2 usages (2 dynamic)   sabi
    def is_overnight_shift(self):
        if self.end < self.start:
            return True
        return False

    # Asociace směny s opatrovníkem
    caregiver = models.ForeignKey(Caregiver, on_delete=models.CASCADE)
```

Obrázek č. 12 – Třída Shift v models.py (caregiver)

Soubor views.py obsahuje logiku pro správnou funkčnost opatrovnické části webu. Nacházejí se zde funkce, které například pouze načítají objekty z databáze (medical_cards - načítání všech registrovaných klientů při zobrazování lékařských karet) nebo jenom zobrazují html šablonu (příručka a její obsah). Komplexnější kód se vyskytuje například ve funkci shift_schedule, která zobrazuje opatrovníkovi jeho směny. Aby se zobrazovaly pouze časově a datově relevantní směny je zde velmi složitá a propracovaná querry (viz Obrázek č. 13), která objekty vyfiltruje. Poté se zavolá funkce, která všechny irelevantní směny ze samotné databáze smaže – viz Obrázek č. 14.

```python
# Query s relevantními směnami
shifts_to_display = caregiver.shift_set.annotate(
    # is_overnight bude True pokud:
    # 1) Datum směny je včerejší, čas začátku směny je větší než čas konce (přes půlnoc)
    # a čas konce směny je větší než současný čas (ještě neskončila)
    # 2) Datum směny je dnešní a čas začátku směny je větší než čas konce (přes půlnoc)
    # (není potřeba kontrolovat zda skončila, jelikož začala dnes a bude končit určitě zítra)
    is_overnight=ExpressionWrapper(
        (Q(date_of_shift=yesterday) & Q(start__gt=F('end')) & Q(end__gt=now.time())) |
        (Q(date_of_shift=now.date()) & Q(start__gt=F('end'))),
        output_field=BooleanField()
    )

# Načte směny, které odpovídají aspoň jedné z těchto podmínek:
# 1) Datum směny je větší než aktuální datum (směna v budoucnu)
# 2) is_overnight je True (vysvětleno nahoře)
# 3) Směna je dnes a ještě neskončila
).filter(
    Q(date_of_shift__gt=now.date()) |
    Q(is_overnight=True) |
    (Q(date_of_shift=now.date()) & Q(end__gt=now.time()))
).order_by('date_of_shift', 'start')
```

Obrázek č. 13 – Query se směnami ve views.py (caregiver)

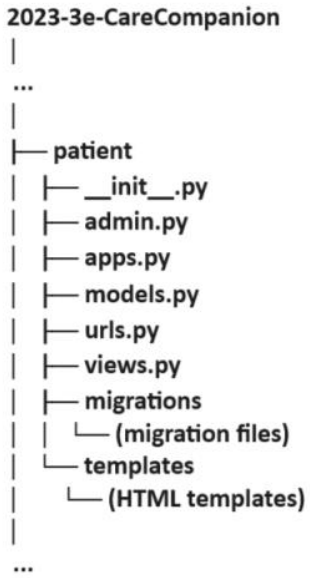
```python
# Funkce, která bere 2 argumenty:
# 'arr_w_all_objs' = pole se všemi objekty
# 'arr_w_upc_objs' = pole pouze s několika objekty z prvního pole
# Je vytvořeno pole 'objs_for_deletion' s objekty, které se nachází v 'arr_w_all_objs' ,ale ne v 'arr_w_upc_objs
# Tato funkce je využita na mazání irelavantních směn a rozvrhů z databáze
2 usages  sabi
def delete_from_database(arr_w_all_objs, arr_w_upc_objs):
    objs_for_deletion = arr_w_all_objs.exclude(id__in=[obj.id for obj in arr_w_upc_objs])
    objs_for_deletion.delete()
```

Obrázek č. 14 – Funkce na mazání objektů z databáze ve views.py (caregiver)

Složka templates obsahuje html šablony, díky kterým se uživatelovi zobrazují informace z backendu a ve složce migrations jsou dva soubory, jedna migrace a druhý __init__.py soubor.

2.3.3 Patient

Poslední aplikace se nazývá patient a představuje klienta. Ten má možnost si zobrazit seznam kde jsou vypsáni všichni opatrovníci a informace k tomu, zda jsou na směně nebo ne, pokud ne, zobrazí se jejich nejbližší směna. Klientské funkce dále zahrnují zobrazení kontaktů na své nejbližší a denní rozvrh (dnešní a budoucí). Grafické zobrazení hierarchie složky aplikace patient je možné vidět na Obrázku č. 15.



Obrázek č. 15 – Hierarchie složky aplikace patient

Soubor urls.py obsahuje pouze inicializace tří webových adres pro zmíněné funkce. V models.py se definují modely Patient, Contact a MedicationIntake s jejich atributy (ty byly popsané v kapitole Databáze). Stejně jako u modelu Caregiver se z přiřazeného uživatelského účtu uloží first_name a last_name jako properties. Také podobně jako u modelu Caregiver (nebylo ovšem zmíněno v kapitole Caregiver), je práce s objekty. V modelové třídě Patient je definovaný atribut object, na který se následně odvolávají modely Contact, MedicationIntake a Activity pomocí vytvořených funkcí ContactManager, MedicationIntakeManager a ActivityManager – viz Obrázek č. 16.

```python
# Pomáha spravovat objekty (v tomto případě contacts, medications, activities)
objects = models.Manager()

sabi
class ContactManager(models.Manager):
    # Funkce pro vytvoření konktaktu
    sabi
    def create_contact(self, patient, relationship, name, phone_number):
        return self.create(patient=patient, relationship=relationship, name=name, phone_number=phone_number)

sabi
class MedicationIntakeManager(models.Manager):
    # Funkce pro vytvoření medikace
    sabi
    def create_medication(self, patient, medication, when, how):
        return self.create(patient=patient, medication=medication, when=when, how=how)

sabi
class ActivityManager(models.Manager):
    # Funkce pro vytvoření rozvrhu
    sabi
    def create_activity(self, patient, date, time, description):
        return self.create(patient=patient, date=date, time=time, description=description)

# Pro manipulaci s objekty MedicationIntake
medications = MedicationIntakeManager()
# Pro manipulaci s objekty Contact
contacts = ContactManager()
# Pro manipulaci s objekty Activity
activities = ActivityManager()
```

Obrázek č. 16 – Práce s objekty ve třídě Patient v models.py (patient)

V souboru views.py jsou funkce, které se zaslouží za správný chod klientských funkcí. V případě načítání opatrovnických směn se postupuje stejným způsobem jako ve views.py v aplikaci Caregiver (viz Obrázek 14, akorát se uloží pouze ta jedna nejdřívější). Při načítání aktivit se dnešní a budoucí ukládají do separátních polí aby se graficky rozlišily.

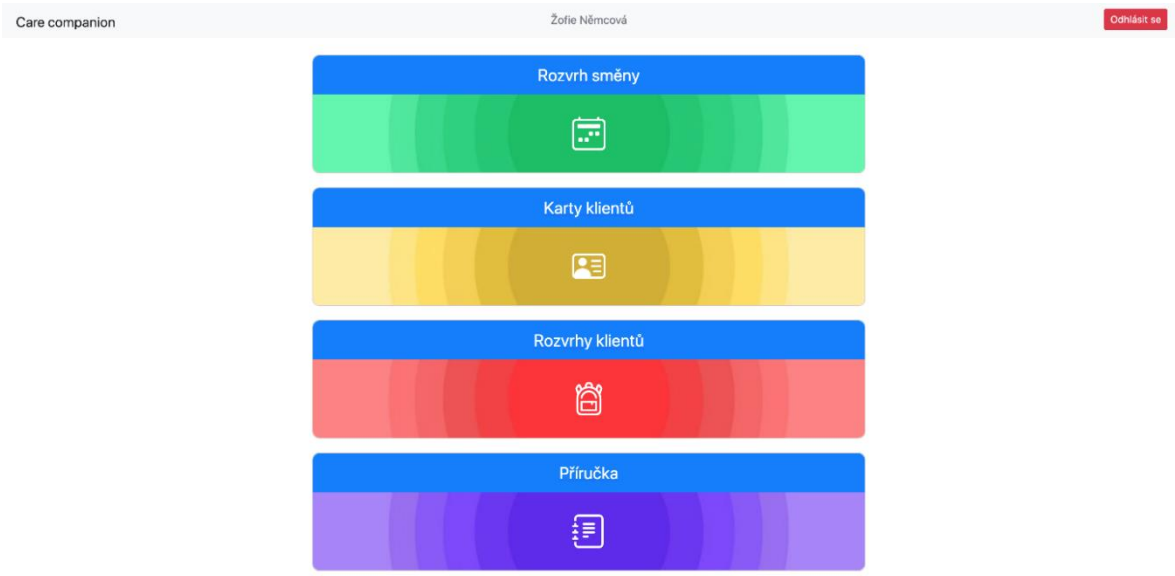
4. Frontend

Frontend slouží k zobrazení všech dat aplikace. Pro výstavbu uživatelského rozhraní jsme se rozhodli použít framework Bootstrap společně s běžným HTML a CSS. Bootstrap byl vybrán pro svoji jednoduchost při implementaci (např. Nevyžaduje API pro komunikaci s backend) a jeho schopnosti jednoduchého vytváření konzistentního designu. Aplikace je určená pro pracovníky v domovech důchodců, ale i pro samotné důchodce – klienty. Z toho důvodu je frontend stavěný tak, aby ikony i všechny pole, na které lze kliknout, byly co největší. Také využíváme různých kontrastních barev pro zjednodušení orientace na stránce.

Aplikace využívá tři typy uživatelských účtů – administrátor, pečovatel a pacient/klient. Každý účet má svoje rozhraní. Administrátorský účet je pro názornost designově oddělen od stránek určených pro běžné uživatele. Stránky pro pečovatele a pacienty vypadají podobně, ale obsahují rozdílné funkce.

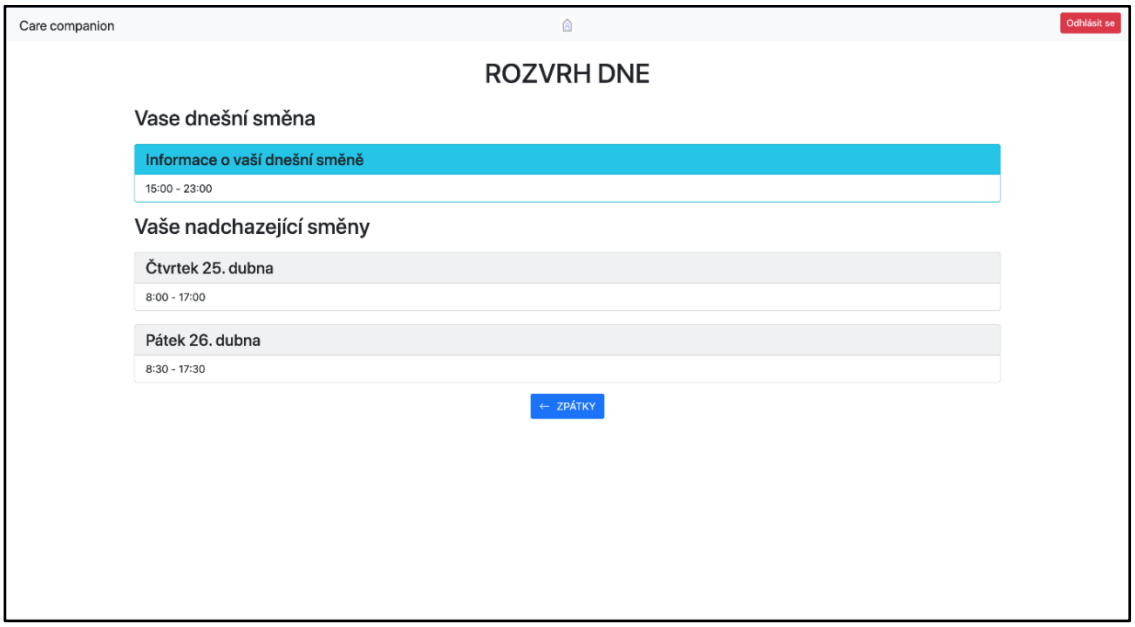
4.1 Účet opatrovníka

Pro jednoduchou orientaci v menu jsme se rozhodli pro systém dlaždic. Pro přesun na danou funkci stačí kliknout kamkoliv na dlaždici. Hlavní menu opatrovníka lze vidět na obrázku č. 17.



Obrázek č. 17 – Menu opatrovníka

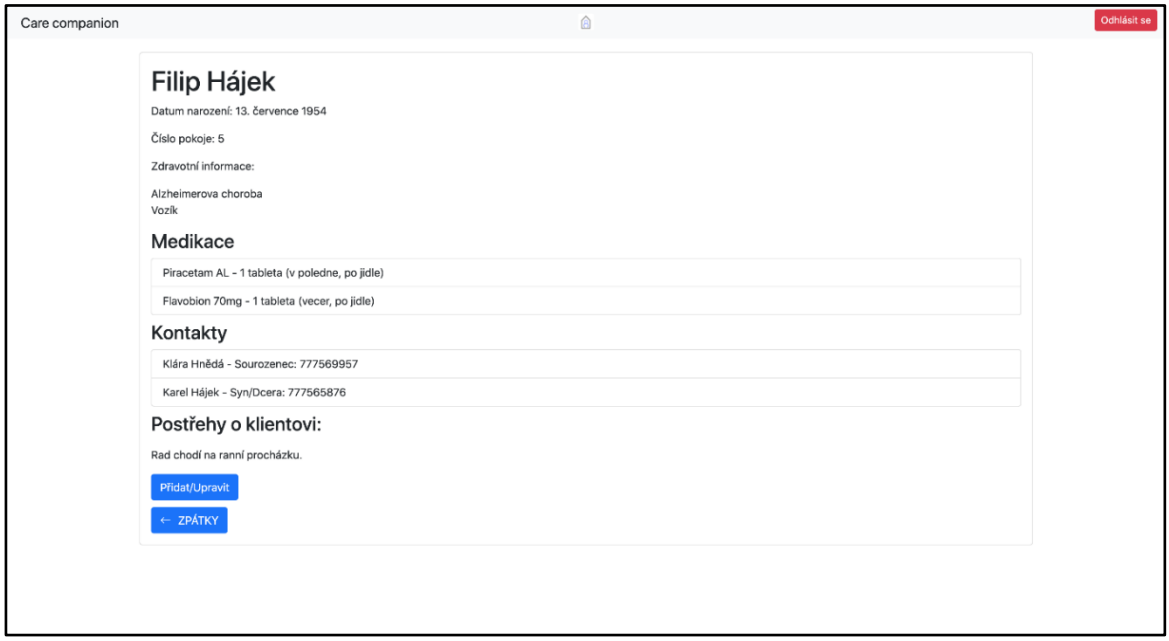
Opatrovník má na výběr z následujících funkcí: rozvrh směny, karty klientů, rozvrhy klientů a příručka. Jak název napovídá v rozvrh směny obsahuje přehled všech budoucích směn opatrovníka. Jeho dnešní směna je zvýrazněna nahoře a následující směny jsou seřazeny chronologicky – viz obrázek č. 18.



Obrázek č. 18 – Směny

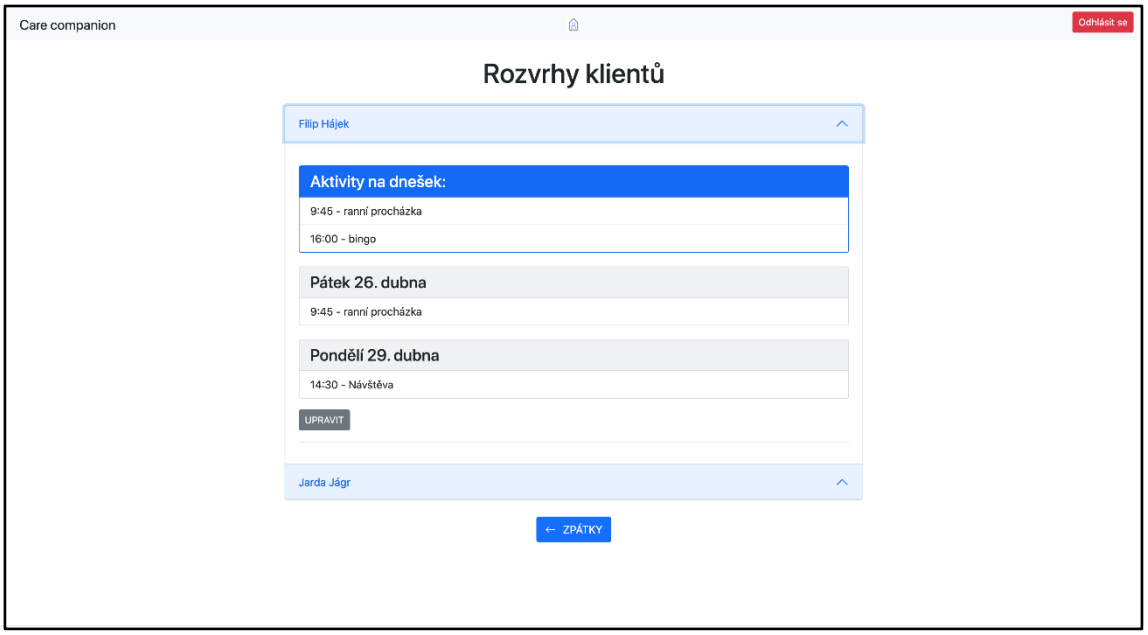
Pomocí karty klientů může opatrovník vyhledávat ze všech pacientů na základě křestního jména, příjmení nebo čísla pokoje. Dále po kliknutí na kolonku s pacientovým jménem vidí kartu pacienta. Karta pacienta obsahuje všechny relevantní informace – jméno, věk, zdravotní informace, medikace a kdy ji aplikovat a kontakty na blízké osoby. Opatrovník má možnost přidávat postřehy o pacientech - např. co má pacient rád nebo další personalizované rady pro jiné opatrovníky. A to dokonce rovnou na kartě bez potřeby znovu načítat stránku – Viz obrázek č. 19 a 20.

Obrázek č. 19 – Vyhledávání karty klienta



Obrázek č. 20 – Karta pacienta

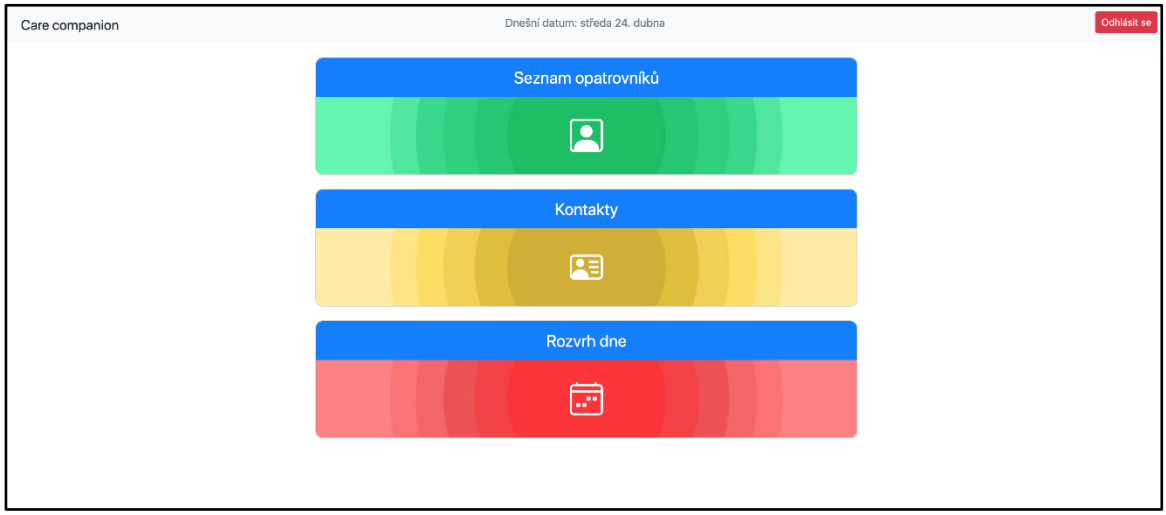
Opatrovníci mají také k dispozici rozvrh klientů. Mají tak větší přehled o tom, co se s klienty děje a ví kam a kdy je potřeba za nimi zajít. Opět mají možnost jednoduše rozvrh pacienta upravovat. Viz obrázek č. 21.



Obrázek č. 21 – Rozvrhy klientů

4.2 Účet klienta

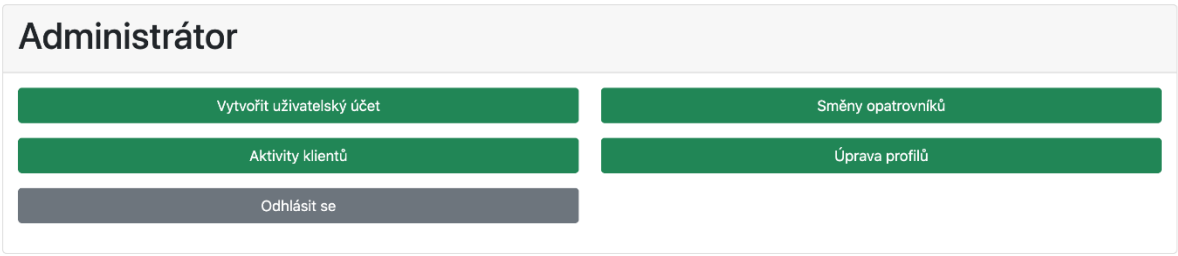
Obdobně má i klient k dispozici několik funkcí – seznam opatrovníků, kontakty a rozvrh dne. Seznam opatrovníků ukazuje, kdy je jaký opatrovník na směně. Pacient má tak přehled o tom, kdo za ním ten den přijde a může se např. těšit na svého oblíbeného ošetřovatele. V kontaktech najde pacient přehled svých kontaktů, tedy telefonní čísla na všechny své nejbližší, které uvedl při příjmu. Díky rozvrhu dne ví pacient přesně co ho čeká ten den i dny následující a ví tak, co má očekávat. Na obrázku č. 22 je zachyceno hlavní menu klienta.



Obrázek č. 22 – Hlavní menu klienta

4.3 Administrátorský účet

Administrátor může přes své rozhraní zakládat nové účty, organizovat směny opatrovníků, přidávat aktivity klientům a upravovat veškeré informace o kterémkoliv z uživatelů. Hlavní menu administrátora je vidět na obrázku č. 23.



Obrázek č. 23 – Administrátorský účet

4.4 Responzivita a DRY

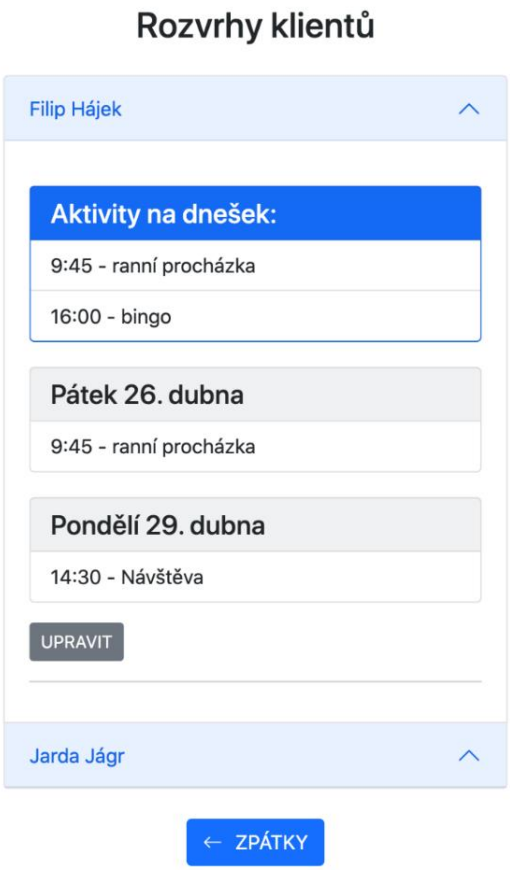
V našem projektu jsme využili schopností Boostrapu a udělali webovou aplikaci plně responzivní. Kladli jsme důraz na mobile-first architekturu. Abychom postupovali podle principu DRY (Don't Repeat Yourself), využili jsme schopnost backend framworku Django dědit *templates*. Do samostatného souboru můžeme např. umístit *navbar* a pokud ho bude potřeba upravit, můžeme tak učinit z jednoho místa a nemusíme upravovat každou stránku jednotlivě.

Na obrázku č. 24 můžeme červeným zvýrazněním vidět, kde jsme využili dědičnost frameworku Django.

```
<body>

<div class="container-fluid bg-light pt-1 pb-1">
  <div class="row">
      <div class="col d-none d-md-block"> <!-- First column, hidden on mobile -->
          <div class="p-1 ml-3 mt-1 navbar-brand">Care companion</div>
      </div>
      <div class="col-9 col-sm-4 text-left text-sm-center"> <!-- Second column -->
          <div class="p-1 mt-1 mb-1 pl-3 pl-sm-0 text-muted">{% block navbar %}{% endblock %}</div>
      </div>
      <div class="col-3 col-sm-4 text-right"> <!-- Third column -->
          <div class="p-1 mr-2 text-end"><a class="btn btn-danger btn-sm" href="{% url 'logout_user' %}">Odhlásit se</a></div>
      </div>
  </div>
</div>

<!-- display caregivers tiles -->
<div class="block-content">
 {% block content %}
 {% endblock %}
</div>
```

Obrázek č. 24 – Dědičnost

Na obrázku č. 25 můžeme vidět, že web je plně responzivní.



Obrázek č. 25 – Screenshot webu z iphone 14 – rezponzivita

6. Závěr

Ve výsledku aplikace splňuje všechny předem stanovené cíle i víc. Obsahuje spoustu užitečných funkcí, které usnadňují každodenní život opatrovníkům i klientům. Je intuitivní na používání, vizuálně přívětivá a responzivní na mobilních zařízeních. Další možné vylepšení by mohlo zahrnovat mini-hru pro klientskou část webu na trénování kognitivnich funkcí a paměti.

Zdroje

[1] Mozilla Corporation, Django introduction, dostupné z https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction navštíveno: 25. 4. 2024

[2] SQLite corporation, features of SQLite, dostupné z https://sqlite.org/features.html navštíveno: 25. 4. 2024

Seznam obrázků