



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Rába Tamás

MARKET PLACE FEJLESZTÉSE SPORTOLÓKNAK

KONZULENS

Dr. Simon Balázs

BUDAPEST, 2024

Tartalomjegyzék

Összefoglaló	6
Abstract.....	7
1 Bevezetés	8
2 Felhasznált technológiák	9
2.1 Python	9
2.2 WebSocket	9
2.3 WSGI (Web Server Gateway Interface)	10
2.4 ASGI (Asynchronous Server Gateway Interface).....	10
2.5 Webes keretrendszer általános bemutatása	10
2.6 Django.....	11
2.6.1 MVT modell komponensei és összegzése	11
2.6.2 A Django főbb jellemzői.....	13
2.7 SQLite.....	19
2.7.1 Főbb jellemzői	19
2.7.2 Használata Django-val	19
2.8 HTML	19
2.9 CSS (Cascading Style Sheets)	20
2.10 Bootstrap.....	20
2.10.1 Főbb jellemzői	20
2.10.2 Használata a projektben	21
2.11 JSON	21
2.12 JavaScript.....	21
2.13 AJAX (Asynchronous JavaScript and XML)	21
2.13.1 Főbb jellemzői	22
2.13.2 Működése.....	22
2.13.3 Használata a projektben	22
3 Tervezés	23
3.1 Specifikáció	23
3.2 MVT alkalmazása	23
3.3 Fájlrendszer.....	24
3.4 Felhasználók	25

3.4.1 Felhasználók típusai.....	25
3.4.2 User és Profile összerendelése	26
3.4.3 Score modell	27
3.5 Hirdetések	28
3.5.1 Sport modell.....	28
3.5.2 ItemCondition modell	28
3.5.3 Item modell	28
3.6 Beszélgetések (chatek).....	29
3.6.1 Chat modell.....	30
3.6.2 ChatMessage modell.....	30
3.7 Adatbázis	31
3.8 Felhasználói felület	32
4 Megvalósítás	34
4.1 Navbar.....	34
4.1.1 Funkciók és elemek	34
4.2 Felhasználói autentikáció és kezelés.....	35
4.2.1 Regisztráció	35
4.2.2 Bejelentkezés	37
4.2.3 Elfelejtett jelszó	38
4.2.4 Kijelentkezés.....	39
4.3 Home page	40
4.3.1 Általános áttekintés.....	40
4.3.2 Home page megvalósítása	41
4.4 Home, About és Advertisers oldalak tetején megjelenő képek	44
4.4.1 Megvalósítás a backenden	44
4.4.2 Megvalósítás a frontenden	45
4.5 About page	45
4.6 Hirdetés részletes oldala	45
4.6.1 Megvalósítás részletei.....	46
4.7 Hirdetés kezelő (CRUD) oldalai	48
4.7.1 My Ads oldal	48
4.7.2 Új hirdetés létrehozása oldal.....	49
4.7.3 Hirdetés szerkesztő felülete	50
4.8 My Profile oldal	51

4.8.1 Jelszó változtatás.....	51
4.9 Chat.....	52
4.9.1 WebSocket alapú chat elvei.....	52
4.9.2 Konfiguráció és beállítások	52
4.9.3 Consumer osztályok.....	53
4.9.4 Routing.....	55
4.9.5 Chat backend logikája.....	55
4.9.6 Chat frontend logikája, My Chats oldal.....	56
4.10 Advertisers page	57
4.11 Adminisztrátor felülete	58
5 Tesztelés	59
5.1 Pytest.....	59
5.2 Coverage	59
5.3 Tesztelés folyamata.....	59
5.3.1 Bemutatott tesztesetek	60
5.4 Lefedettség.....	62
6 Összefoglalás.....	63
Irodalomjegyzék.....	64

HALLGATÓI NYILATKOZAT

Alulírott **Rába Tamás**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot/diplomatervet **(nem kívánt törlendő)** meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2024. 12. 05.

.....
Rába Tamás

Összefoglaló

Napjaink online szolgáltatásai egyre inkább a virtuális közösségekre és a felhasználók közötti interakcióra építenek, különösen olyan platformokon, ahol a közösség tagjai megoszthatják, cserélhetik vagy értékesíthetik már nem használt tárgyaikat. Az ilyen típusú online piactereket gyakran market place-eknek nevezik, és rendkívül népszerűek mindennapi életünk számos területén.

Ugyanakkor ezek a szolgáltatások sokszor széles, általános közönséget céloznak meg, ami megnehezíti a releváns találatok és célzott ajánlatok biztosítását a felhasználók számára. Ez a probléma különösen érzékenyen érinti a speciális igényű közösségeket, például a sportolók csoportját, akik egyedi igényeikhez illeszkedő termékeket keresnek.

A szakdolgozatom célja egy olyan market place kialakítása, amely kifejezetten sportolóknak szól. A diplomamunkám egy Python Django alapú webalkalmazás fejlesztésének folyamatát mutatja be.

A dolgozat első részében a projekt során felhasznált technológiákat ismertetem, bemutatva azok szerepét és hozzájárulását a fejlesztés különböző szakaszaikhoz. A második rész az alkalmazás tervezési folyamatát, megvalósítását és tesztelését mutatja be, részletesen kitérek az egyes fejlesztési lépésekre, a funkciók megvalósítására, valamint a felhasználói élményre.

Abstract

Today's online services increasingly rely on virtual communities and interactions among users, especially on platforms where members can share, exchange, or sell items they no longer need. These types of online marketplaces have become highly popular in various aspects of daily life.

However, many of these services are designed for a wide, general audience, which can make it challenging to provide relevant results and targeted offers for users. This issue particularly affects niche communities, such as athletes, who seek products tailored to their unique needs.

The purpose of my thesis is to create a marketplace specifically for athletes. My thesis details the development process of a web application built on Python and Django, aimed at fulfilling these specific needs.

The first part of the thesis describes the technologies used throughout the project, explaining their roles and contributions across different stages of development. The second part covers the design, implementation, and testing of the application, detailing each development step, the implementation of various functions, and considerations for enhancing user experience.

1 Bevezetés

A webalkalmazások napjainkban az informatika egyik legmeghatározóbb eszközei, mivel platformfüggetlen működésük révén széles körben elérhetők. Ezek az alkalmazások lehetőséget nyújtanak a felhasználóknak, hogy internetkapcsolaton keresztül bárhol és bármikor hozzáférjenek az általuk használt szolgáltatásokhoz, miközben folyamatos frissítésekkel és valós idejű adatszinkronizációval biztosítják a naprakész funkcionalitást.

A market place egy különösen népszerű webalkalmazás-típus, amely az eladók és vásárlók találkozási pontjaként szolgál.

A megvalósítandó sportolóknak szóló market place egy specifikus, célzott közösséget szolgál ki, amely további előnyöket nyújt az általános piacterekhez képest. Egy ilyen platform lehetőséget biztosít a sportolóknak arra, hogy könnyen és gyorsan beszerezzék a sporteszközöket, vagy ruházatot, miközben lehetőséget teremt a már nem használt tárgyak újraértékesítésére is.

Szakdolgozatom egy ilyen market place megtervezésének, megvalósításának és tesztelésének folyamatát mutatja be. Az alkalmazás lehetővé teszi a látogatók számára, hogy különféle sporteszközök és hirdetőik között keressenek, hirdetőket értékeljenek, valamint kapcsolatba lépjenek egymással.

A következő fejezetben az alkalmazás megvalósításához használt technológiákat ismertetem. A 3. fejezetben a tervezési folyamat lépéseit mutatom be, mely szükséges a market place megvalósításához. A 4. fejezet a webalkalmazás megvalósításának részleteit mutatom be a felhasználói felületen keresztül. Az 5. fejezetben bemutatom, hogyan teszteltem az alkalmazást, hogy a funkciónak megfelelő működése garantált legyen. A 6. fejezetben összegzem az elvégzett munkát és a lehetséges továbbfejlesztési lehetőségeket.

2 Felhasznált technológiák

2.1 Python

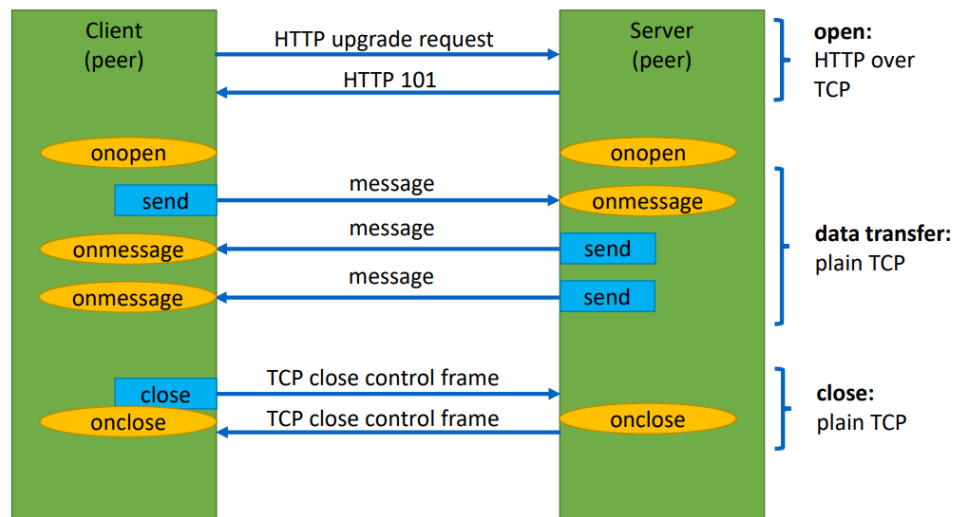
A Python egy magas szintű, értelmezett, és dinamikus típusú programozási nyelv. Pythonra jellemző az olvashatóságra helyezett nagy hangsúly és a tiszta szintaxis, amely lehetővé teszi, hogy a fejlesztők gyorsan és hatékonyan dolgozzanak. A nyelv célja a kód olvashatóságának biztosítása a minimalista szintaxison keresztül, és támogatja az objektumorientált, funkcionális és procedurális programozási paradigmákat is, így rugalmasan alkalmazható különböző fejlesztési területeken.

A Python kiterjedt és folyamatosan bővülő standard könyvtára lehetővé teszi az alapvető fájlkezelést, adatfeldolgozást, hálózati kommunikációt és adatbázis-kezelést. Ezen felül számos, harmadik féltől származó könyvtár érhető el, amelyek segítségével a Python különösen népszerűvé vált a webfejlesztésben, a gépi tanulásban, az adatfeldolgozásban és a tudományos kutatásban [1][2].

2.2 WebSocket

A WebSocket egy olyan kommunikációs protokoll, amely teljes duplex kapcsolatot tesz lehetővé a kliens és a szerver között egyetlen TCP kapcsolat használatával. Ez azt jelenti, hogy a kliens és a szerver bármikor küldhet üzeneteket egymásnak, anélkül hogy új kapcsolatot kellene létrehozni minden egyes üzenetváltáshoz. A WebSocket ideális valós idejű alkalmazásokhoz, mint például chat alkalmazások, élő frissítést igénylő dashboardok, játékok vagy bármilyen alkalmazás, ahol a kommunikációban a késleltetés csökkenthető.

A WebSocket kapcsolat létrehozásához a kliens egy speciális HTTP kérést küld a szervernek, amelyben jelzi, hogy WebSocket kapcsolatot szeretne létrehozni. Ha a szerver elfogadja a kérést, a HTTP kapcsolat WebSocket kapcsolattá alakul. Innentől kezdve a kommunikáció kétirányú, és a felek üzeneteket küldhetnek egymásnak a kapcsolat lezárásáig [3]. A kommunikációt szemlélteti az 1. ábra.



1. Ábra: WebSocket protokoll [3]

2.3 WSGI (Web Server Gateway Interface)

A WSGI a Python webalkalmazások egyik alapvető interfésze, amely a webkiszolgáló és az alkalmazás közötti kommunikációért felelős. A WSGI lineáris és szinkron működésre tervezett architektúrája egyszerű, mégis robusztus megoldást nyújt a klasszikus, nem valós idejű webalkalmazások kiszolgálására [4].

2.4 ASGI (Asynchronous Server Gateway Interface)

Az ASGI a WSGI modern, aszinkron működésre továbbfejlesztett változata, amely a webalkalmazások kiszolgálására szolgál. Az ASGI támogatja az olyan új generációs webes technológiákat, mint a WebSocket-ek vagy a hosszú élettartamú HTTP-kapcsolatok, miközben visszafelé kompatibilitást biztosít a hagyományos WSGI-alapú alkalmazásokkal. Az ASGI szabvány rugalmasságot és skálázhatóságot kínál, amely különösen fontos a valós idejű funkciókat nyújtó alkalmazások esetén [5].

2.5 Webes keretrendszer általános bemutatása

A webes keretrendszerek célja, hogy megkönnyítsék a webalkalmazások fejlesztését és kezelését azáltal, hogy sablonokat, előre definiált komponenseket és szabványokat biztosítanak a fejlesztők számára. Ezek a keretrendszerek lehetővé teszik, hogy a fejlesztők újrahasznosítható kódokat használjanak, így kevesebb időt kell fordítaniuk az ismétlődő feladatokra, mint például az autentikációra, az adatbázis-kezelésre vagy a felhasználói felületek kialakítására.

A webes keretrendszerek két fő típusa:

- **Frontend keretrendszerek (például React, Angular, Vue.js):** Ezek a felhasználói felületek létrehozására szolgálnak, dinamikus és interaktív elemek megjelenítésére. A frontend keretrendszerek gyakran JavaScript alapúak, és lehetővé teszik az aszinkron adatkezelést, illetve a felhasználói élmény javítását.
- **Backend keretrendszerek (például Django, Flask, Ruby on Rails, Express):** Ezek a szerveroldali logika kezelésére és az adatbázisokkal való interakció megkönnyítésére szolgálnak. A backend keretrendszerek olyan funkciókat biztosítanak, mint az URL-kezelés, az adatok feldolgozása, a biztonság és az autentikáció.

A megfelelő keretrendszer kiválasztása a fejlesztési követelményektől függ, és a keretrendszerek által nyújtott funkcionalitások jelentős időmegtakarítást eredményezhetnek a fejlesztési folyamat során [6].

2.6 Django

A Django egy Python alapú, nyílt forráskódú webes keretrendszer, amely kifejezetten gyors, biztonságos és skálázható webalkalmazások készítésére lett kifejlesztve. 2005-ben jelent meg, és azóta az egyik legnépszerűbb backend keretrendszerré vált a Python közösségen belül. Django az „MVT” (Model-View-Template) architektúrát követi, amely hasonló a klasszikus MVC (Model-View-Controller) mintához, de némileg eltér a névhasználatban.

Django különösen előnyös azoknak a fejlesztőknek, akik gyorsan és hatékonyan szeretnének biztonságos webalkalmazásokat készíteni. Az olyan funkciók, mint az adminisztrációs felület, az autentikáció és az ORM, jelentősen csökkentik a fejlesztési időt és növelik a kód minőségét [7][8].

2.6.1 MVT modell komponensei és összegzése

2.6.1.1 Model

Az adatbázis struktúráját képviseli, és a Django ORM-mel együttműködve biztosítja az adatok tárolását és lekérdezését. A modellekben definiálhatók az adatmezők, valamint az üzleti logika egyes elemei, például a validálás és az adatfeldolgozás [9].

2.6.1.2 Template

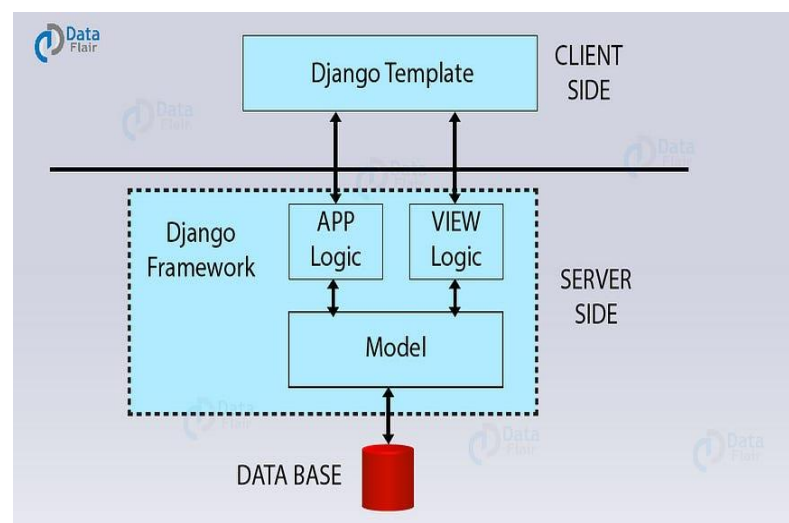
A sablonrendszer felelős a HTML struktúra kialakításáért és a dinamikus tartalmak megjelenítéséért. A sablonokban változókat, ciklusokat és feltételes logikát is használhatunk, így biztosítva, hogy a megjelenítendő adatok rugalmasan és strukturáltan legyenek elérhetők. A sablonok lehetővé teszik a logika elkülönítését a megjelenítéstől, ami átláthatóbbá teszi a kódot és könnyebb módosíthatóságot eredményez [10].

2.6.1.3 View

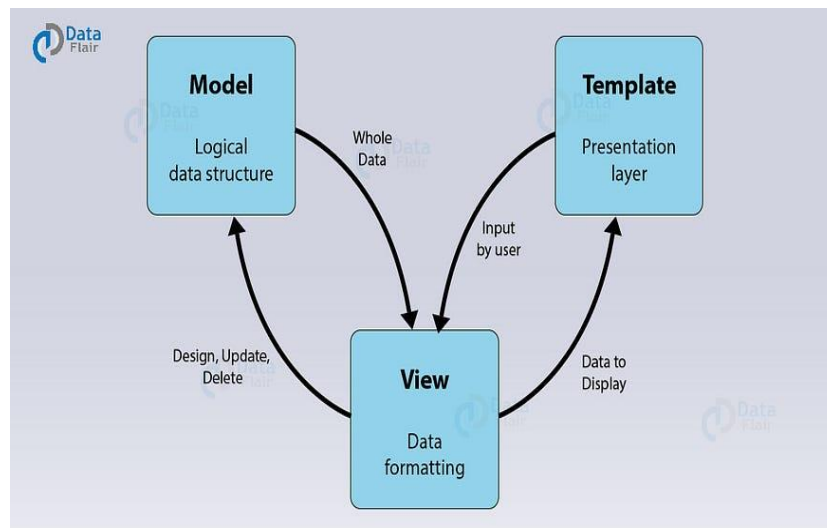
A nézetek kezelik az alkalmazás üzleti logikáját és a beérkező kéréseket. Meghatározzák, hogy egy adott kérésre milyen adatokat kell lekérdezni, és milyen sablonban jelenjenek meg. Két fő típusa létezik: a függvényalapú (function-based) és az osztályalapú (class-based) nézetek. Az osztályalapú nézetek, például, strukturáltabb megközelítést kínálnak, mivel lehetőséget adnak újrahasznosítható nézetek megírására [11].

2.6.1.4 Az MVT modell összegzése

Az MVT modell szerkezete világos és jól szervezett keretrendszert biztosít, amely lehetővé teszi a kód különböző rétegeinek elválasztását, és elősegíti a clean kódírást. Django-ban a modellek, sablonok és nézetek szoros együttműködése biztosítja, hogy az alkalmazások gyorsan és hatékonyan épüljenek fel, miközben a komplexitás is jól kezelhető. A 2. ábra szemlélteti a Django architektúráját, a 3. ábra pedig a MVT felépítését mutatja be.



2. Ábra: Django architektúrája [12]



3. Ábra: MVT felépítése [12]

2.6.2 A Django főbb jellemzői

2.6.2.1 Automatizált adminisztrációs felület

A Django automatikusan generál egy adminisztrációs felületet a modellek alapján, amely lehetővé teszi az adatok kezelését az alkalmazás adminisztrátorai számára. Az adminisztrációs felület számos előre beépített funkcióval rendelkezik, mint például:

- **CRUD műveletek (Create, Read, Update, Delete):** Lehetővé teszi az adatok hozzáadását, szerkesztését, megtekintését és törlését közvetlenül az admin felületen keresztül.
- **Keresés és szűrés:** Az adminisztrációs felület támogatja a keresési és szűrési lehetőségeket, így könnyen kezelhetők a nagyméretű adattáblák.
- **Testreszabhatóság:** Az adminisztrációs felület egyedileg testre szabható, így megadhatók olyan saját mezők és funkciók, amelyek az adott alkalmazás igényeihez igazodnak.
- **Biztonsági beállítások és jogosultságkezelés:** Az adminisztrátorok és a szerkesztők különböző jogosultságokkal kezelhetők, így biztonságos környezetet biztosít az érzékeny adatok kezeléséhez.

Ezzel az automatikusan generált felülettel a fejlesztők rengeteg időt takaríthatnak meg, és lehetőséget kapnak az adminisztrációs feladatok gyors elvégzésére anélkül, hogy külön UI-t kellene fejleszteniük [13].

2.6.2.2 Beépített hitelesítési rendszer

A Django natív támogatást nyújt a felhasználók hitelesítésére és jogosultságkezelésére, amely biztosítja az adatok biztonságát és a megfelelő jogosultsági szintek alkalmazását. A rendszer alapértelmezetten tartalmazza:

- **Felhasználókezelés:** Beépített funkciókat kínál a felhasználók létrehozására, módosítására és törlésére.
- **Bejelentkezés és kijelentkezés:** Kész nézetek és URL-ek biztosítják a felhasználók be- és kijelentkezését, valamint a jelszóváltoztatás és a jelszó-visszaállítás lehetőségét.
- **Csoport- és jogosultságkezelés:** Lehetővé teszi a felhasználók csoportokba rendezését és a jogosultsági szintek hozzárendelését, így a rendszeren belüli hozzáférési szabályok könnyen alkalmazhatók.
- **Több hitelesítési mechanizmus támogatása:** Alapértelmezés szerint támogatja a jelszó alapú hitelesítést, de más mechanizmusok, mint például az egyedi hitelesítési backendek (OAuth, LDAP stb.) is integrálhatók.
- **Testreszabhatóság:** A hitelesítési rendszer könnyen bővíthető és testre szabható, így lehetőség nyílik saját modellek és autentikációs logika megvalósítására.

A beépített hitelesítési és jogosultságkezelési rendszer megkönnyíti a fejlesztők munkáját, miközben biztosítja az alkalmazás biztonságát és felhasználói hozzáférési szabályainak betartását [14].

2.6.2.3 ORM

A Django ORM egy hatékony eszköz, amely lehetővé teszi, hogy a fejlesztők adatbázis-interakciókat Python objektumokon keresztül valósítsanak meg, elrejtve az SQL lekérdezések bonyolultságát. Az ORM biztosítja, hogy az adatbázisokban lévő adatok Python osztályokkal és objektumokkal legyenek kezelhetők, ezáltal leegyszerűsítve az adatkezelést és javítva a kód olvashatóságát és karbantarthatóságát.

Főbb jellemzői:

- **Modellek használata:**

A Django ORM alapját a modellek alkotják, amelyek Python osztályokként definiálhatók. Minden modell megfelel egy adatbázis-táblának, és a modell mezői az adatbázis oszlopainak felelnek meg. Például:

```
from django.db import models
class Jatekos(models.Model):
    nev = models.CharField(max_length=100)
    kor = models.IntegerField()
    csapat = models.ForeignKey('Csapat',
                               on_delete=models.CASCADE)
```

Ez a Jatekos modell egy táblát képvisel, ahol a nev és kor mezők az oszlopok, a csapat mező pedig egy idegen kulcs kapcsolatot jelöl a Csapat táblához [9].

- **Automatizált adatbázis műveletek:**

Django ORM-je automatikusan generálja az SQL lekérdezéseket a CRUD (Create, Read, Update, Delete) műveletekhez, ami lehetővé teszi, hogy a fejlesztők közvetlenül Python parancsokkal végezzenek adatbázis műveleteket [15]. Például egy új Jatekos objektum létrehozása és mentése az adatbázisba:

```
jatekos = Jatekos(nev="Kiss Péter", kor=25)
jatekos.save()
```

- **Kapcsolatok kezelése:**

Az ORM lehetővé teszi különböző adatbázis-táblák közötti kapcsolatok kezelését, mint például egy-egy (OneToOneField), egy-több (ForeignKey), és több-több (ManyToManyField) kapcsolatokat. Ezekkel az ORM közvetlenül tudja kezelni a kapcsolatokban lévő adatokat [16].

- **Szűrések és rendezések:**

Az ORM segítségével egyszerűen szűrhetők és rendezhetők az adatbázis-lekérdezések. Például az összes 25 évnél idősebb sportoló lekérdezése:

```
idosebb_jatekosok = Jatekos.objects.filter(kor__gt=25)
```

- **Tranzakciók és adat-integritás:**

Az ORM automatikusan kezeli az adatbázis tranzakciókat, biztosítva az adatbázis integritását és konzisztenciáját. Például, ha egy `save()` művelet nem sikerül, az ORM automatikusan visszavonja az adatbázis tranzakciót [17].

- **Migrations (Migrációk):**

A Django ORM tartalmaz egy migrációs rendszert, amely lehetővé teszi az adatbázis sémaváltozások kezelését anélkül, hogy SQL parancsokat kellene írni. A fejlesztők létrehozhatják és alkalmazhatják a migrációkat, amikor új modelleket adnak hozzá vagy módosítják a meglévőket [18].

A Django ORM segítségével a fejlesztők egységesen dolgozhatnak különböző adatbázisokkal (pl. PostgreSQL, MySQL, SQLite), mivel a Django automatikusan optimalizálja a műveleteket az adott adatbázis-rendszerhez. Az ORM által támogatott magas szintű absztrakció lehetővé teszi az adatbázis-szintű műveletek gyors és biztonságos megvalósítását.

A Django ORM e tulajdonságai révén a fejlesztési folyamat gyorsabb, egyszerűbb, és könnyebben karbantartható.

2.6.2.4 Sablonrendszer

A Django beépített sablonrendszere lehetővé teszi, hogy dinamikus tartalmat HTML-ben jelenítsünk meg, ami különösen hasznos az interaktív webalkalmazások fejlesztése során. A Django sablonrendszere támogatja a **változók** használatát, **ciklusokat** és **feltételeket**. Így a sablonokban könnyedén megjeleníthetők az adatbázisból származó vagy más módon generált adatok anélkül, hogy közvetlen Python kódot kellene írni a HTML-ben.

A rendszer **kétirányú elválasztást** biztosít az adat és a megjelenítés között, ami azt jelenti, hogy az alkalmazás üzleti logikája elkülöníthető a felhasználói felülettől. Ezáltal a sablonok átláthatóbbá és karbantarthatóbbá válnak.

Django sablonok főbb funkciói:

- **Változók használata:**

A sablonokban egyszerűen lehet változókat megjeleníteni, például: `{{ user.username }}`, amely kiírja az aktuális felhasználó nevét.

- **Logikai szerkezetek:**

A for ciklus és az if feltétel kezelése közvetlenül a sablonban is lehetséges, például:

```
<ul>
    {% for item in items %}
        <li>{{ item.name }}</li>
    {% endfor %}
</ul>
```

Ez a kód egy listát jelenít meg az items lista elemeivel. Ha az items üres, a for ciklus nem jelenít meg semmit.

- **Blokkok és öröklődés:**

A sablonrendszer támogatja az öröklődést, ami lehetővé teszi, hogy egy alapértelmezett (pl. főoldali) sablonból más sablonok származzanak. Ezáltal egyszerűbb a közös elemek, például a navigációs menük és a láblécek kezelése:

```
<html>
<body>
    <header>{% block header %}{% endblock %}</header>
    <main>{% block content %}{% endblock %}</main>
    <footer>{% block footer %}{% endblock %}</footer>
</body>
</html>
```

Az egyes aloldalak csak a content blokkot töltik fel egyedi tartalommal, így könnyen lehet egységes megjelenést biztosítani az egész webalkalmazás számára.

További eszközök és kiterjesztések:

- **Szűrők:** A Django sablonokban elérhető beépített szűrők, például a date és length, lehetővé teszik az adatok formázását. Például: `{{ date_variable|date:"Y-m-d" }}`.
- **Külső bővítmények:** A fejlesztők egyedi sablon szűrőket és tageket is létrehozhatnak, így a Django templating rendszere rendkívül rugalmasan bővíthető.

A Django sablonrendszere tehát átlátható, könnyen bővíthető megoldást nyújt az adatok megjelenítésére, miközben elkülöníti az üzleti logikát és a prezentációs réteget [10][19].

2.6.2.5 Django Channels

A Django Channels egy kiegészítő a Django-hoz, amely lehetővé teszi a WebSocket és más, valós idejű protokollok használatát. Alapvetően kiterjeszti a Django hagyományos, szinkron működését aszinkron alapokra, amely különösen hasznos chat, értesítések vagy valós idejű események feldolgozása esetén [20][21][22].

A Django Channels főbb jellemzői:

- **WebSocket támogatás:** WebSocketeken keresztül lehetővé teszi a kétirányú, valós idejű kommunikációt, amely ideális a chat funkció megvalósításához.
- **Aszinkron kezelés:** A Channels támogatja az aszinkron műveleteket, amelyek jelentősen növelik az alkalmazás teljesítményét és lehetőséget biztosítanak skálázódásra.
- **Message Queues és Background Tasks:** A Channels lehetőséget biztosít üzenetsorok és háttérfolyamatok kezelésére, így az alkalmazás képes nagy mennyiségű valós idejű adat kezelésére is.

2.6.2.6 Daphne

A Daphne az ASGI szerverek egyik kiemelkedő megvalósítása, amelyet a Django Channels projekt részeként fejlesztettek ki. Az aszinkron működésre tervezett Daphne lehetővé teszi, hogy a webalkalmazások valós idejű kommunikációs funkciókat, például WebSocket-alapú adatcserét is támogassanak. Az egyszerű telepíthetőségének és kiváló teljesítményének köszönhetően a Daphne ideális választás azokhoz a projektekhez, amelyek az aszinkron kommunikáció előnyeit szeretnék kiaknázni [23].

2.6.2.7 Kiterjedt közösség és dokumentáció

A Django keretrendszer egyik kiemelkedő erőssége a mögötte álló, folyamatosan bővülő közösség és a kiváló dokumentáció. A Django projekt nyílt forráskódú, így a közösség aktívan hozzájárul új funkciók, hibajavítások és bővítmények fejlesztéséhez. Ennek köszönhetően a keretrendszer rendszeresen kap

frissítéseket, amelyek új funkciókat, biztonsági javításokat és teljesítménybeli optimalizálásokat tartalmaznak.

A Django dokumentációja részletes, jól szervezett és kezdőktől a haladó fejlesztőkig mindenki számára érthető [24].

2.7 SQLite

Az SQLite egy könnyűsúlyú, beágyazott SQL adatbázis-kezelő, amely nem igényel külön szerveret vagy telepítést, mivel minden adatot egyetlen fájlban tárol. Ez különösen előnyös a kisebb webalkalmazások vagy mobilalkalmazások esetében, ahol nincs szükség bonyolult, többfelhasználós adatbázis-infrastruktúrára [25].

2.7.1 Főbb jellemzői

- **Könnyű használat:** Az SQLite nem igényel külön adatbázis-szerveret, minden adatot közvetlenül egy fájlba ír, így egyszerűbb a telepítése és kezelése.
- **Teljes SQL támogatás:** Támogatja az SQL-92 szabványt, így a legtöbb szabványos SQL utasítással kompatibilis.
- **Alacsony erőforrásigény:** Kis méretének köszönhetően rendkívül gyors, és minimális memóriaigénnyel rendelkezik.
- **Ideális fejlesztéshez és teszteléshez:** Django alapértelmezett adatbázisa, ami lehetővé teszi a gyors fejlesztést és a könnyű tesztelést.

2.7.2 Használata Django-val

A Django alapértelmezésben az SQLite adatbázist használja, így a fejlesztők válláról levéve egy adatbázis telepítésének a terhét. Az SQLite fájl alapú adatbázisként jól működik a Django ORM-mel, amely segítségével Python objektumokként kezelhetők az adatbázisok adatai.

Az SQLite azonban nem skálázható olyan jól, mint a nagyobb adatbázisok (pl. PostgreSQL, MySQL), ezért inkább kisebb projektekhez vagy lokális fejlesztési környezetben ajánlott használni [26].

2.8 HTML

A HTML (HyperText Markup Language) a web alapvető jelölőnyelve, amelyet a weboldalak struktúrájának és tartalmának meghatározására használnak. A HTML

segítségével meghatározható, hogy milyen elemek jelenjenek meg egy weboldalon, és hogyan rendezzék el ezeket az elemeket.

A HTML dokumentumok hierarchikus struktúrában épülnek fel, ahol az elemek egymásba ágyazhatók. Ez a fa struktúra határozza meg a weboldal DOM (Document Object Model) szerkezetét [27].

2.9 CSS (Cascading Style Sheets)

A CSS egy stílusleíró nyelv, amelyet a HTML elemek megjelenésének szabályozására használnak. A CSS segítségével meghatározhatjuk az elemek színeit, méreteit, elrendezését, és számos egyéb vizuális tulajdonságot. Alapvető szerepe van a weboldalak stílusának és vizuális megjelenésének kialakításában [28][29].

2.10 Bootstrap

A Bootstrap egy nyílt forráskódú CSS keretrendszer, amelyet a Twitter fejlesztett ki, hogy gyorsan és könnyen lehessen reszponzív, modern weboldalakat készíteni. A Bootstrap olyan előre definiált stílusokat és összetevőket biztosít, amelyek minimalizálják a kézi CSS-kódolást, és lehetővé teszik a webalkalmazások egységes, professzionális megjelenését [30].

2.10.1 Főbb jellemzői

- **Reszponzív tervezés:** A Bootstrap "mobile-first" megközelítést alkalmaz, amely lehetővé teszi, hogy a weboldal minden eszközön jól nézzen ki, a kisebb kijelzőktől egészen a nagyobb képernyőig.
- **Rácsrendszer (Grid System):** A Bootstrap rácsos elrendezési rendszere (Grid) könnyen kezelhetővé teszi a weboldal különböző elemeinek elhelyezését és méretezését. Ez a rendszer 12 oszlopból áll, és egyszerű CSS osztályokkal testreszabható.
- **UI-összetevők:** A Bootstrap számos beépített összetevőt biztosít, mint például gombok, navigációs menük, űrlapok és még sok más, amelyek előre definiált stílusokkal és interakciókkal rendelkeznek.

- **Testreszabható:** A Bootstrap egy alaposan dokumentált CSS keretrendszer, amely könnyen testreszabható saját színek, betűtípusok és elrendezési szabályok segítségével.

2.10.2 Használata a projektben

A projektben a Bootstrapet használtam a felhasználói felület gyors és egységes kialakításához. A különféle UI-összetevők, például a gombok, űrlapok és navigációs menük segítettek a webalkalmazás esztétikus és könnyen használható felületének kialakításában.

2.11 JSON

A JSON (JavaScript Object Notation) egy könnyen olvasható, szöveges adatformátum, amelyet adatcsere céljából használnak. Hierarchikus szerkezetű, így alkalmas összetett adatok tárolására és szervezésére. A JSON az adatokat kulcs-érték párok formájában tárolja, amelyek egymáshoz kapcsolódva képesek strukturált adatot, például objektumokat és tömböket létrehozni. Ez a formátum különösen népszerű a webfejlesztésben, hiszen számos programozási nyelv (köztük a Python és a JavaScript) natív támogatást biztosít hozzá [31].

Django alkalmazásokban a JSON-t gyakran használjuk az adatok kliens és szerver közötti átvitelére.

2.12 JavaScript

A JavaScript egy kliensoldali programozási nyelv, amely lehetővé teszi a HTML struktúra dinamikus változtatását, továbbá adatok manipulálását. Így lehetőséget ad interaktív elemek hozzáadására weboldalakhoz, mint például a gombok, űrlapok, animációk és egyéb dinamikus tartalmak. A JavaScript támogatja a WebSocket technológiát, így a munkám során valós idejű események, adatok és üzenetek megjelenítésére tudtam használni [32].

2.13 AJAX (Asynchronous JavaScript and XML)

Az AJAX egy olyan technológia, amely lehetővé teszi a weboldalak számára az aszinkron adatátvitelt a kliens és a szerver között anélkül, hogy az egész oldalt újratöltené. Az AJAX technológia segítségével a felhasználók számára gyorsabb és zökkenőmentesebb élményt lehet biztosítani, hiszen az oldalelemek frissítése egyes

részekre korlátozódik, ami csökkenti a teljes oldal újrenderelésének szükségességét [33][34].

2.13.1 Főbb jellemzői

- **Aszinkron adatátvitel:** Az AJAX lehetővé teszi a háttérben történő adatlekérdezést, amely nem akadályozza a felhasználó interakcióit a felülettel.
- **Interaktivitás növelése:** Az AJAX segítségével interaktívabbá válik az alkalmazás, hiszen lehetővé teszi például a real-time üzenetküldést, a chat- és értékelőrendszerek dinamikus frissítését.
- **JSON támogatás:** Az adatátvitel legtöbbször JSON formátumban történik, amely könnyen használható JavaScript-ben és kisebb méretű, mint a korábban elterjedt XML.

2.13.2 Működése

Az AJAX művelet során egy JavaScript XMLHttpRequest objektumot használ, amely kapcsolatot hoz létre a kliens és a szerver között. Ez a kapcsolat lehetővé teszi a szükséges adatok lekérését vagy továbbítását a szerverre. Az eredményként kapott adat JavaScript segítségével kerül feldolgozásra, amely lehetővé teszi az oldalelemek dinamikus frissítését.

2.13.3 Használata a projektben

A projekt során az AJAX technológiát alkalmaztam a chat funkció megvalósításánál, ahol fontos volt a gyors és zökkenőmentes adatátvitel. A chat üzenetek betöltése AJAX-alapú hívásokkal történik, lehetővé téve a felhasználók számára, hogy a beszélgetések között váltani tudjanak anélkül, hogy az oldal folyamatosan újra töltődne.

3 Tervezés

3.1 Specifikáció

Feladatom egy olyan **sportolók** számára készült **market place** webalkalmazás létrehozása volt, amely egyszerű, modern felhasználói felülettel és egyértelmű funkciókkal rendelkezik. Az alkalmazás lehetőséget nyújt a látogatók számára, hogy különféle sporteszközök és hirdetőik között keressenek, hirdetőket értékeljenek, valamint kapcsolatba lépjenek egymással. A rendszer különböző jogosultsági szintekkel kínál hozzáférést a felhasználóknak.

Összefoglalva a feladatom **3 komponensre** osztható: **felhasználók** és kezelésük, **hirdetések** és kezelésük, **beszélgetések** és kezelésük.

3.2 MVT alkalmazása

A Python alapú Django keretrendszerre épülő market place alkalmazás tervezése során az MVT architektúra bizonyult a legalkalmasabbnak. Az MVT modell segítségével az alkalmazás különálló rétegekre osztható, amelyek egymástól függetlenül fejleszthetők és karbantarthatók, miközben biztosítják az adatok, a logika és a megjelenítés közötti együttműködést.

A Django MVT rendszerében a modellek felelősek az alkalmazás adatainak kezeléséért és az adatbázis-struktúra reprezentációjáért, míg a view-k az üzleti logikát valósítják meg, amely a kliens kéréseinek feldolgozását és a megfelelő válaszok előállítását végzi. A template-ek az adatok megjelenítésére szolgálnak, lehetővé téve a dinamikus tartalom renderelését az alkalmazás felhasználói felületén.

A tervezés következő alfejezeteiben részletesen bemutatásra kerülnek az alkalmazásban használt modellek, amelyek az adatstruktúrákat és az üzleti szabályok alapjait képezik. Ezek az alfejezetek az adatbázis alapú elemek közötti kapcsolatok tervezésére és az egyes entitások felelősségi köreinek meghatározására helyezik a hangsúlyt. A megvalósítás fejezet alfejezetei ezzel szemben a view-kra és a template-ekre fókuszálnak, bemutatva a dinamikus tartalom előállítását és a felhasználói élményt meghatározó megoldásokat.

3.3 Fájlrendszer

A Python Django projektek fájlrendszere logikusan szervezett struktúrával rendelkezik, amely biztosítja a könnyű karbantarthatóságot és az egyes funkciók átlátható elkülönítését. Az alábbiakban röviden leírom a legfontosabb fájlok szerepét és funkcióját, amelyek a market place alkalmazás fejlesztése során meghatározóak.

- **models.py:** Ez a fájl tartalmazza az adatbázis modelljeinek definícióit, amelyek az alkalmazás adatstruktúráját és a kapcsolódó üzleti logikát írják le. A modellek felelnek az adatok validálásáért és az adatbázis-műveletekért.
- **views.py:** A nézetek logikáját megvalósító fájl, amely a felhasználók kéréseit feldolgozza, és a megfelelő válaszokat, például HTML-t vagy JSON-t generál. Ez a réteg közvetíti az adatokat a modellek és a megjelenítés között.
- **urls.py:** Az alkalmazás URL-rendszerét határozza meg, összekötve az egyes URL-eket a megfelelő nézetekkel. Ez a fájl biztosítja a felhasználói kérések pontos kezelését és az alkalmazás navigációs struktúráját.
- **forms.py:** Az űrlapok kezeléséért felelős fájl, amely segíti az adatok validálását és egyszerűsíti az űrlapok megjelenítését. A Django beépített formakezelési lehetőségeivel kombinálva minimalizálja a kódismétlést.
- **templates:** Az alkalmazás megjelenítési rétegét tartalmazó könyvtárstruktúra, amelyben a HTML alapú sablonok helyezkednek el. Ezek dinamikusan renderelhetők a nézetek által biztosított adatokkal, biztosítva a felhasználói felület testreszabhatóságát.
- **admin.py:** Az adminisztrációs felület testreszabásáért és konfigurációjáért felelős fájl, amely lehetővé teszi a modellek könnyű kezelését a beépített adminisztrációs rendszerben.
- **apps.py:** Az alkalmazás konfigurációs beállításait tartalmazza, biztosítva az alkalmazás megfelelő regisztrációját a Django projektben.

A WebSocket-alapú megvalósításhoz kapcsolódó további fájlok:

- **consumers.py:** Ez a fájl kezeli a WebSocket kapcsolatok logikáját, lehetővé téve a valós idejű kommunikációt a szerver és a kliens között. A fogyasztók

(consumerek) felelősek a WebSocket események feldolgozásáért, például az üzenetek fogadásáért és küldéséért.

- **routing.py**: A WebSocket URL-ek kezelését végzi, hasonlóan az urls.py fájlhoz, de kifejezetten a WebSocket kapcsolatokra fókuszálva. Ez határozza meg, hogy melyik WebSocket esemény melyik fogyasztóhoz tartozik.

3.4 Felhasználók

3.4.1 Felhasználók típusai

A rendszerben három felhasználói típus került megkülönböztetésre. Az első a nem bejelentkezett látogató, aki csak böngészésre és alapvető információk megtekintésére jogosult. A második a regisztrált felhasználó, aki már hozzáfér a funkciók többségéhez. A harmadik az adminisztrátor szerepkörrel rendelkező felhasználó, aki a rendszer teljeskörű felügyeletét látja el.

3.4.1.1 Látogató felhasználó

A látogató felhasználó a webalkalmazás főoldalán böngészhet a market place hirdetései között, ahol keresési, szűrési és rendezési lehetőségeket is talál. Megtekintheti az egyes hirdetések részletes információit. Az About oldalon tájékozódhat a FitTrade nevű platformról, míg az Advertisers oldalon a hirdetőik között kereshet, rendezhet, valamint megtekintheti a hirdetőik értékeléseit, rövid bemutatkozásukat és az általuk meghirdetett termékeket.

Ugyanakkor a látogató felhasználónak nincs lehetősége kapcsolatfelvételre vagy beszélgetésre a hirdetőikkel. Nem jogosult hirdetéseket feladni, azokat módosítani vagy kezelni, és nem adhat értékelést a hirdetőkről az Advertisers oldalon.

3.4.1.2 Regisztrált felhasználó

A regisztrált és bejelentkezett felhasználók számára a látogatói funkciók teljes körűen elérhetők. Ezen túlmenően lehetőséget kapnak arra, hogy kapcsolatba lépjenek a hirdetőikkel és beszélgetéseket folytassanak velük. A regisztrált felhasználók saját hirdetéseket adhatnak fel, amelyeket később módosíthatnak és kezelhetnek, illetve beszélgetéseket bonyolíthatnak le a hirdetéseikkel kapcsolatban. Emellett értékelést is adhatnak a hirdetőkről az Advertisers oldalon.

3.4.1.3 Adminisztrátor

Az adminisztrációs szerepkör a Django adminisztrációs felületén keresztül érhető el, amelyet a Django keretrendszer automatikusan biztosít. Az adminisztrátor kezelheti a hirdetéseket, felhasználókat, felhasználók közötti beszélgetéseket és a hirdetőik értékelését.

Fontos megjegyezni, hogy az adminisztrátorok **nem férnek hozzá a publikus bejelentkezési felülethez**. Az adminisztrátori feladatok kizárólag a dedikált adminisztrációs felületen keresztül végezhetők el, amely erős biztonsági védelmet nyújt az illetéktelen hozzáférés ellen.

3.4.2 User és Profile összerendelése

A felhasználók kezelésére a Django keretrendszer beépített User modellje került felhasználására, amelyet egy Profile modell egészít ki. Ez lehetővé teszi a felhasználói fiókokhoz kapcsolódó további adatok tárolását anélkül, hogy módosítani kellene a Django alaprendszerét. Az összerendelés a Profile modellben a OneToOneField mezővel valósult meg, amely garantálja, hogy minden felhasználóhoz pontosan egy profil tartozik.

A Django beépített **User** modelljéből az alábbi alapértelmezett mezők kerültek felhasználásra:

- **username**: Egyedi azonosító a felhasználó számára.
- **first_name**: A felhasználó keresztnéve.
- **last_name**: A felhasználó vezetéknéve.
- **email**: A felhasználó e-mail címe.
- **password**: A felhasználó jelszava (titkosítva tárolva).

A Profile modell felhasználásával a következő mezőkkel bővítettem az alapértelmezett User modellt:

- **bio**: A felhasználó bemutatkozását tárolja.
- **dateofbirth**: A felhasználó születési dátumát tartalmazza.

A Profile mezői lehetővé teszik, hogy a felhasználók további személyes adatokat is megadhassanak, amelyeket a market place platformon való interakciók során használhatnak.

Ez a megoldás több szempontból is előnyös:

- A **User** modell változatlan maradhat, így kihasználható a Django alapértelmezett autentikációs és jogosultságkezelési rendszere.
- A **Profile** modellben tárolt adatok könnyen bővíthetők, például további mezőkkel.
- A **OneToOneField** kapcsolattal biztosítható a jól strukturált és átlátható adatmodell, elkerülve az adatok redundanciáját.

Ez a megközelítés biztosítja a projekt rugalmasságát és skálázhatóságát, miközben egyszerűsíti a felhasználói adatok kezelését.

3.4.3 Score modell

A Score modell a felhasználók közötti értékelési rendszer alapját képezi, lehetővé téve, hogy a regisztrált felhasználók egymást értékeljék. Ez a funkció visszajelzést nyújt a hirdetők megbízhatóságáról. **Főbb mezők:**

- `rated_user`: Az a felhasználó, akit az értékelés érint. Ez egy `ForeignKey` mező, amely kapcsolatot teremt a Django beépített `User` modelljével.
- `rater_user`: Az a felhasználó, aki az értékelést készítette. Szintén `ForeignKey` mezővel van összekapcsolva a `User` modellel, külön `related_name` attribútummal a könnyebb hivatkozás érdekében.
- `score`: Az értékelés értéke, amely egy egész számként kerül tárolásra.
- `created_at`: Az értékelés létrehozásának időpontját automatikusan rögzíti a rendszer.

3.4.3.1 Egyediségi korlátozás

Annak érdekében, hogy egy felhasználó csak egyszer értékelhessen egy másik felhasználót - ezzel megelőzve a redundáns vagy többszöri értékeléseket - kényszerek kerültek felhasználásra.

3.5 Hirdetések

A Hirdetések fejezet célja, hogy bemutassa, hogyan valósul meg a webalkalmazásban a hirdetések modellezése és kezelése. A hirdetéseket három fő modell valósítja meg: Sport, ItemCondition és Item. Ezek a modellek együtt biztosítják, hogy a hirdetések megfelelő struktúrában kerüljenek tárolásra, és lehetővé teszik a hirdetések különféle tulajdonságainak egyszerű kezelését.

3.5.1 Sport modell

Ez a funkció nemcsak a hirdetést feladó felhasználónak nyújt kényelmet azáltal, hogy meghatározhatja, mely sportághoz kapcsolódik az általa kínált eszköz vagy termék, hanem a market place-re látogató érdeklődők számára is egyszerűbbé teszi a keresést és böngészést nagy mennyiségű hirdetések között.

Az elérhető sportág opciókat az adminisztrátor kezelheti a rendszer adminisztrációs felületén. Az adminisztrátor jogosult új sportágakat hozzáadni, meglévőket módosítani vagy törölni, így a kategóriák mindig naprakészen tarthatók, igazodva a felhasználói igényekhez.

3.5.2 ItemCondition modell

A hirdetések állapotának nyilvántartása kulcsfontosságú ahhoz, hogy a felhasználók pontos információt adhassanak meg az általuk kínált sporteszközök állapotáról. Az ItemCondition modell lehetőséget nyújt arra, hogy egy-egy hirdetésnél meghatározható legyen a termék állapota, például új, jó állapotú vagy esetleg sérült. A hirdetés állapotok kezelése ugyanúgy az adminisztrátor szerepköréhez tartozik.

3.5.3 Item modell

A legösszetettebb modell az Item, amely az alkalmazásban megjelenő hirdetések minden releváns adatát tartalmazza.

A hirdetéseket reprezentáló **Item** mezői:

- **name:** A hirdetések egyértelmű azonosítását a name mező biztosítja.
- **sport:** A modell egy sport mezőt is tartalmaz, amely idegen kulcs formájában kapcsolódik a Sport modellhez, így minden hirdetéshez hozzárendelhető egy konkrét sportág.

- **advertiser:** A hirdetések tulajdonosát az advertiser mező azonosítja, amely szintén idegen kulcs, és a felhasználóhoz kapcsolódó Profile modellhez kötődik. Ez a kapcsolat biztosítja, hogy a hirdetések mindig egy adott felhasználóhoz legyenek rendelve, aki felelős azok kezeléséért.
- **description:** A description mező lehetőséget ad arra, hogy részletes leírást adjanak a meghirdetett sporteszközökről a hirdető, de üresen is hagyhatják. Ez a leírás tartalmazhat például technikai specifikációkat, a használatának jellemzőit, vagy akár egyedi tulajdonságokat, amelyek megkülönböztetik azt más hasonló termékektől.
- **price:** A price mező kötelezően kitöltendő, és a hirdetés árát tárolja.
- **createdate:** Az adatbázisban rögzített létrehozási dátumot a createdate mező tartalmazza, amely opcionális, de megadható.
- **sold:** A hirdetések értékesítési állapotát a sold mező tárolja, amely egy logikai érték formájában jelzi, hogy az adott hirdetés már elkelt-e.
- **image:** A rendszer lehetőséget biztosít a hirdetésekhez képek társítására is, amelyet az image mező kezel. Az alapértelmezett kép a "noimage.png" fájl, ha a felhasználó nem tölt fel egyéni képet.
- **condition:** A hirdetés állapotának meghatározását az condition mező biztosítja, amely idegen kulcsként kapcsolódik az ItemCondition modellhez.

A hirdetések modellezése átgondolt és strukturált módon valósul meg, amely biztosítja a rendszer funkcionalitásának egyszerű bővíthetőségét és karbantarthatóságát.

3.6 Beszélgetések (chatek)

A market place hatékony működésének egyik alapvető eleme a felhasználók közötti kommunikáció lehetősége, amelyet a rendszer beszélgetések formájában valósít meg. A beszélgetések célja, hogy a hirdető és az érdeklődő egyszerűen és átláthatóan egyeztethessenek az adott hirdetésekről.

3.6.1 Chat modell

A Chat modell biztosítja az előző bekezdésben tárgyaltakhoz szükséges adatstruktúrát. Ez a modell az alábbi főbb mezőket tartalmazza:

- **participant1 és participant2:** A két mező a beszélgetésben résztvevő felhasználókat reprezentálja. Ezek biztosítják, hogy a rendszer mindig pontosan azonosítani tudja a beszélgetés feleit. A mezők az alkalmazás felhasználói modelljére mutatnak, ami lehetővé teszi a dinamikus kapcsolatot a felhasználói adatokkal.
- **item:** Ez a mező a beszélgetés tárgyát képező hirdetést tárolja, amelyre vonatkozóan a kommunikáció folyik. Az *item* mező a hirdetéseket reprezentáló modellre mutat, így könnyedén kapcsolható a beszélgetés a releváns hirdetéshez.
- **lastupdated:** Ez a mező a beszélgetés utolsó módosításának időpontját rögzíti. Automatikusan frissül, amikor a beszélgetéshez új üzenetet adnak hozzá, így a későbbiek során a beszélgetések sorrendezhetők lesznek.

3.6.1.1 Egyediség kényszer

Az adatbázisban biztosítani kell, hogy egy adott hirdetéshez egy adott felhasználói páros között csak egyetlen beszélgetés létezhesen. Ezt a következő kényszer valósítja meg:

```
UniqueConstraint(fields=["participant1", "participant2",  
                        "item"], name="participants_item_unique_together")
```

Ez a kényszer garantálja, hogy ugyanaz a két felhasználó ugyanazon hirdetéshez ne hozhasson létre több beszélgetést. Az egyediség a participant1, participant2 és item mezők kombinációjára vonatkozik.

3.6.2 ChatMessage modell

A beszélgetésekhez kapcsolódó üzenetek kezelését a ChatMessage modell végzi, amely az adott beszélgetéshez tartozó egyedi üzeneteket tárolja.

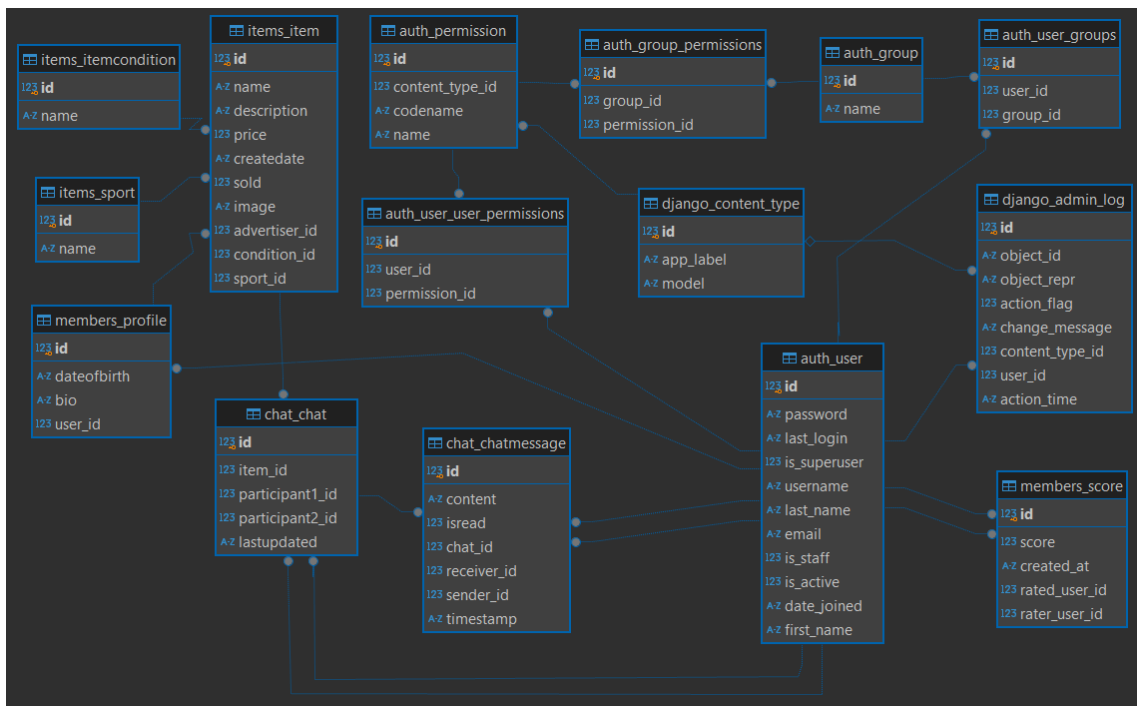
A modell az alábbi mezőket tartalmazza:

- **chat:** Ez a mező az adott üzenethez tartozó beszélgetést azonosítja. A beszélgetések és az üzenetek közötti logikai kapcsolatot biztosítja, ami elengedhetetlen az üzenetek rendszerezéséhez és visszakereséséhez.
- **sender és receiver:** Ezek a mezők az üzenetet küldő és fogadó felhasználókat reprezentálják. Az alkalmazás felhasználói modelljére mutatnak, biztosítva, hogy az üzenet mindig pontosan azonosítható legyen a kommunikációs felek között.
- **content:** Ez a mező az üzenet szöveges tartalmát tárolja. Itt kerül rögzítésre az a szöveg, amelyet a felhasználók egymásnak küldenek, ezzel biztosítva a kommunikáció lényegének megőrzését.
- **timestamp:** Az üzenet küldésének időpontját tárolja, lehetővé téve az üzenetek időbeli sorrendjének pontos meghatározását.
- **isread:** Ez a logikai mező az üzenet olvasottsági állapotát rögzíti. Segítségével követhető, hogy az adott üzenet elolvasásra került-e, ami fontos információ lehet a felhasználók számára.

3.7 Adatbázis

A webalkalmazás adatbázisát az SQLite hozza létre, amely egy könnyűsúlyú, beágyazott adatbázismotor. Az adatbázis kezelése a Django ORM segítségével történik, amely megkönnyíti az adatokkal való munkát, hiszen az adatbázis-műveletek Python osztályok és objektumok formájában valósíthatók meg.

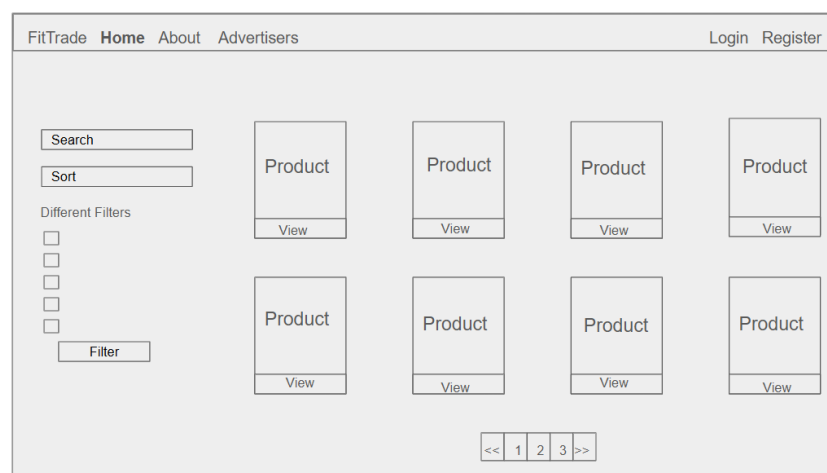
Az alábbi ábra szemlélteti az adatbázis szerkezetét, amely tartalmazza az alkalmazáshoz tartozó egyedi modellek tábláit, valamint a Django által automatikusan generált táblákat. A 4. ábra segítségével könnyen átlátható, hogyan kapcsolódnak egymáshoz a különböző táblák.



4. Ábra: Adatbázisdiagram

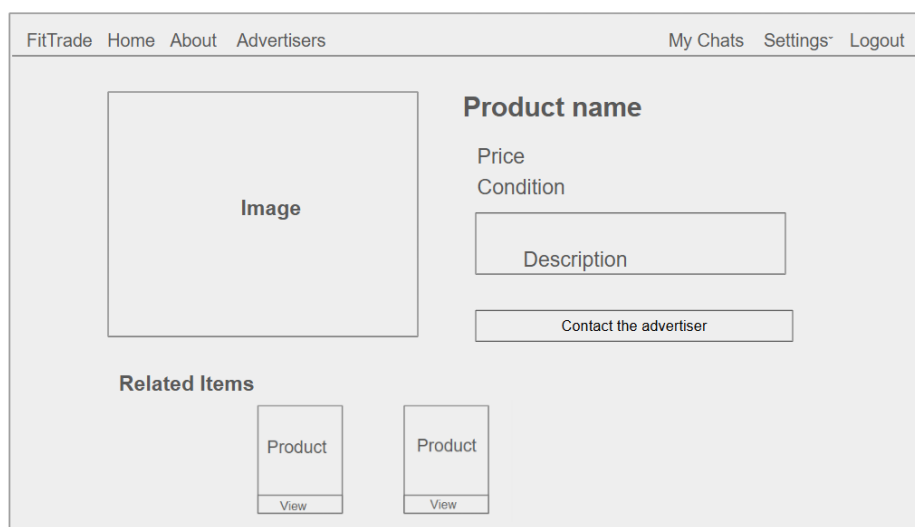
3.8 Felhasználói felület

A felület főbb oldalait vázlatyszerű rajzok szemléltetik, amelyek az oldal struktúráját és funkcióit is bemutatják. Az 5. ábra például a főoldal (Home page) tervezett kinézetét mutatja, ahol a termékek kártyaelrendezésben jelennek meg. Egy oldalon nyolc termék látható, az oldal alján lapozási lehetőség segíti a további termékek böngészését. Az oldalsó bal sávban található a keresési, szűrési és rendezési funkciók, amelyek megkönnyítik a felhasználók számára a releváns hirdetések gyors megtalálását.



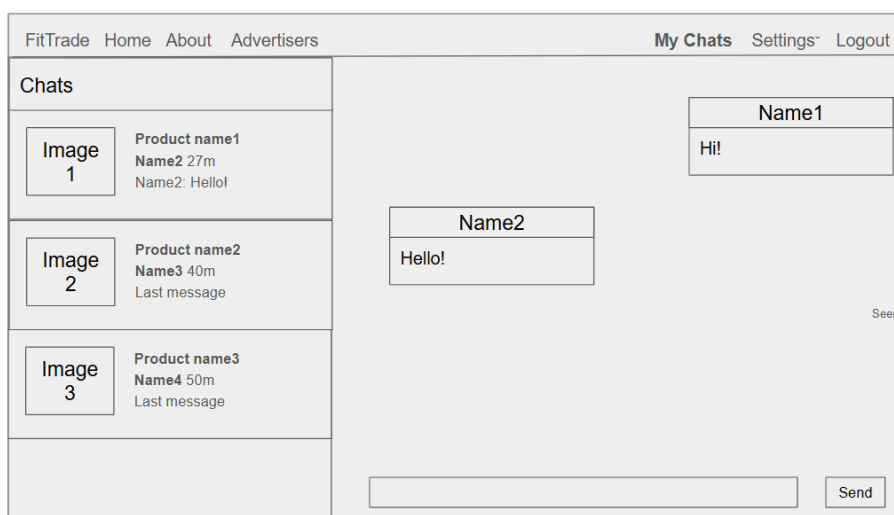
5. Ábra: Home page vázlatyszerű rajza

A 6. ábra egy konkrét hirdetés részletes megjelenítésére szolgáló oldalt ábrázol, amelyen az adott termék minden fontos információja szerepel. További funkcióként a bejelentkezett felhasználók számára egy gomb biztosítja, hogy azonnal új, üres csevegést indíthassanak a hirdetővel. Emellett az oldalon kapcsolódó hirdetések is megjelennek, amelyek további releváns termékeket ajánlanak a látogatóknak.



6. Ábra: Hirdetés részletes oldalának vázlat szerű rajza

A 7. ábra a csevegési felület (chat) kinézetét mutatja be, amely a hasonló market place alkalmazások jól ismert mintáit követi. A bal oldali sávban a csevegések görgethető listája jelenik meg, amelyben alapvető információk segítik a navigációt, míg a képernyő fennmaradó része a kiválasztott beszélgetés részleteit jeleníti meg.



7. Ábra: My Chats oldal vázlat szerű rajza

4 Megvalósítás

Ez a fejezet a kész market place-t a felhasználói felület bemutatásán keresztül ismerteti. Részletesen kitér azokra a funkciókra és megoldásokra, amelyek technológiai szempontból érdekesek lehetnek az olvasó számára.

4.1 Navbar

A market place webalkalmazás navigációs sávja, azaz a navbar, kulcsszerepet játszik a felhasználói élményben, hiszen biztosítja a gyors és intuitív elérést az oldal legfontosabb funkcióihoz. Letisztult megjelenése és funkcionalitása a modern webalkalmazások alapvető követelményeinek felel meg, miközben dinamikusan alkalmazkodik a felhasználói státuszhoz és az eszköz méretéhez.

A navbar fejlesztése során a következő technológiák kerültek alkalmazásra:

- **HTML:** Biztosítja a struktúrát és a statikus tartalom elrendezését.
- **CSS és Bootstrap:** Használatuk teszi a megjelenést reszponzívvá, különösen az olyan Bootstrap-osztályok, mint a navbar-expand-lg (reszponzív navigáció), bg-light (világos háttér) és dropdown-menu-end (igazítás a lenyíló menünél).
- **Django sablonnyelv:** Teszi lehetővé a dinamikus tartalom, például a bejelentkezési állapotnak megfelelő funkciók megjelenítését `{% if user.is_authenticated %}` feltételekkel.

4.1.1 Funkciók és elemek

A navbar elemei egyértelműen elkülöníthetők, hogy a különböző felhasználói igényeket kiszolgálják. A sáv bal oldalán található a FitTrade logó, amely minden oldalon elérhető és a kezdőlapra navigál. Az ezt követő fő menüpontok – Home, About, és Advertisers – lehetővé teszik a legfontosabb oldalak közötti gyors váltást.

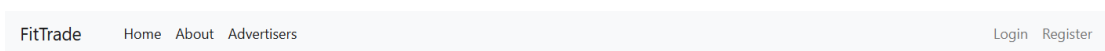
A jobb oldalon található elemek a felhasználói hitelesítési állapothoz igazodnak. Bejelentkezett felhasználók számára elérhetők a személyes funkciók, például a My Chats (a csevegések listázására), illetve a lenyíló menüben található My Profile (profiloldal) és My Ads (hirdetések). Ugyanitt elhelyezésre került a Logout

funkció, amely lehetővé teszi a kijelentkezést. A navbar ezen állapotát az 8. ábra is szemlélteti.

Amennyiben a felhasználó nincs bejelentkezve, a navigációs sáv egyszerűbb tartalmat mutat: a Login (bejelentkezés) és Register (regisztráció) opciók jelennek meg, biztosítva az egyszerű hozzáférést ezekhez az alapvető funkciókhoz. A navbar ezen állapotát az 9. ábra is szemlélteti.



8. Ábra: Navbar bejelentkezett állapotban



9. Ábra: Navbar kijelentkezett állapotban

4.2 Felhasználói autentikáció és kezelés

Ez az alfejezet a felhasználói autentikációval kapcsolatos funkciók megvalósítását mutatja be, beleértve a regisztrációt, a bejelentkezést, a kijelentkezést és az elfelejtett jelszó kezelését.

4.2.1 Regisztráció

A regisztráció folyamata kulcsfontosságú a felhasználói fiókok létrehozása során, hiszen ez biztosítja a rendszerhez való hozzáférést és a személyre szabott profil létrejöttét.

A regisztráció során két különálló űrlap van használatban: az egyik a **UserSignUpForm**, amely a felhasználó alapvető adatait (pl. felhasználónév, email) kezeli, míg a másik a **ProfileSignUpForm**, amely a profilhoz tartozó kiegészítő adatokat (pl. születési dátum, bemutatkozás) kezeli. A regisztrációs oldalon ezek egyszerre jelennek meg és egyszerre is kerülnek beküldésre, mely a 10. ábrán is látható.

FitTrade Home About Advertisers Login Register

Please fill out the registration form

User Name
Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

First Name

Last Name

Email Address

Password

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Confirm Password
Enter the same password as before, for verification.

Date of birth:
 éééé. hh. nn.

Write something about yourself...

Register

Copyright © Market place for Athletes

10. Ábra: Regisztrációs oldal

4.2.1.1 UserSignUpForm

Ez az osztály a `UserCreationForm` kiterjesztésével jött létre, és a Django beépített felhasználói modelljét használja. Az űrlap testreszabott mezőket tartalmaz, amelyek CSS-osztályokkal és helykitöltő szövegekkel vannak ellátva a felhasználói élmény javítása érdekében. A form-ban a mezők megjelenítése a form elemeinek a `widget.attrs` tulajdonságán keresztül kerül beállításra. A `help_text`, segítségsszövegek, amelyek megkönnyítik a kitöltést a felhasználók számára.

4.2.1.2 ProfileSignUpForm

Ez az űrlap a profilmodellhez kapcsolódik, és opcionális mezőket tartalmaz. A mezők megjelenése és típusa itt is testreszabott.

4.2.1.3 A regisztráció logikája

- Érvényesítés:** Ellenőrzi, hogy mindkét űrlap helyesen lett-e kitöltve (`is_valid()`).
- Adatok mentése:** Az űrlapokból nyert adatokat elmenti az adatbázisba. A profil létrehozásakor a felhasználót hozzárendeli a profilhoz.

3. **Visszajelzés:** Sikeres regisztráció esetén üzenetet küld a felhasználónak, és átirányítja a kezdőlapra.

4.2.1.4 Megjelenítés

A `{% csrf_token %}` a regisztrációs űrlap védelmére szolgáló token, amely megakadályozza a CSRF-támadásokat. Az űrlapok a `userform` és `profileform` változókon keresztül érhetők el, amelyeket a `view`-ban adok át, és automatikusan HTML-mezőkké alakulnak a `{{ userform.as_p }}` és `{{ profileform.as_p }}` kódrészletek használatával. A **visszajelző üzenetek** (sikeres vagy hibás kitöltés esetén) a sablonban az üzenetek környezetébe lettek beillesztve.

4.2.2 Bejelentkezés

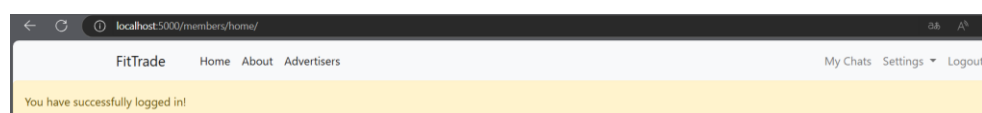
A Bejelentkezés alfejezet célja, hogy bemutassa, miként működik az alkalmazásban a felhasználói fiókokhoz tartozó hitelesítési folyamat, amely lehetővé teszi a regisztrált felhasználók számára a hozzáférést a platform funkcióihoz. A bejelentkezési oldal minimalista és felhasználóbarát kialakítással rendelkezik, ahol a felhasználók a **felhasználónevüket és jelszavukat** megadva hitelesíthetik magukat, melyet a 11. ábra is szemléltet. Az egyszerűség mellett a folyamat biztonsági szempontokat is figyelembe vesz, például CSRF-token használatával védi a kéréseket. A 12. ábrán látható, hogy helyes bejelentkezés után a felhasználó visszairányításra kerül a kezdőlapra és szöveges üzenet tájékoztatja a sikeres bejelentkezésről, míg hibás hitelesítési adatok esetén egy figyelmeztető üzenet tájékoztatja a problémáról.

Please log in

Log in

[Forgot your password?](#)

11. Ábra: Bejelentkezési felület



12. Ábra: Sikeres bejelentkezést követően a Home page-re átirányítás, szöveges üzenet

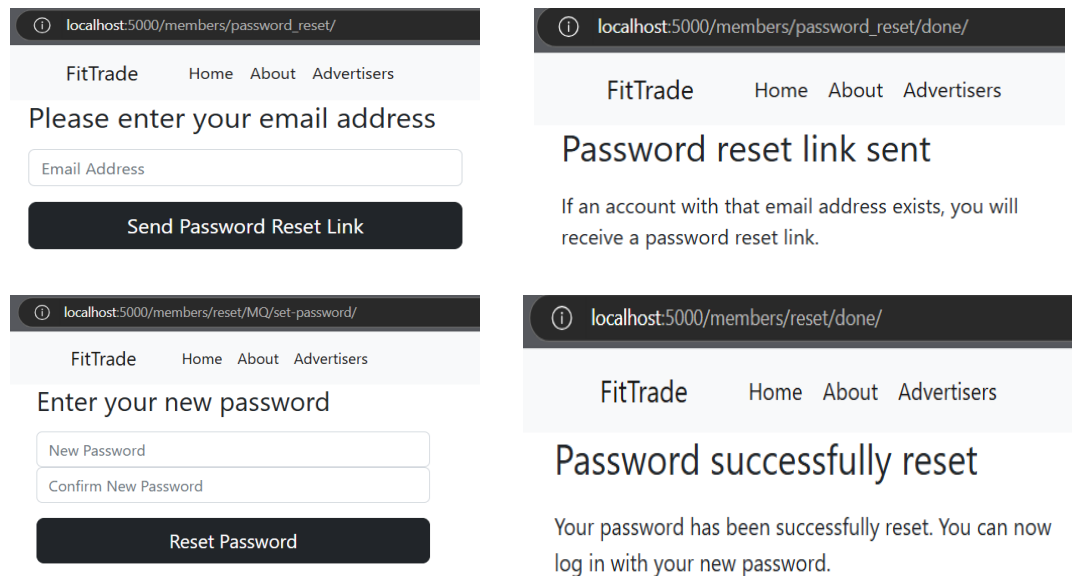
4.2.2.1 Bejelentkezés logikája

- **request.method == "POST"**: Ellenőrzi, hogy a bejelentkezési kérelem HTTP POST típusú-e, ami a hitelesítési adatok továbbítására szolgál.
- **authenticate**: A Django beépített autentikációs függvénye ellenőrzi, hogy az adott felhasználónév és jelszó kombináció érvényes-e.
- **login**: Amennyiben a hitelesítés sikeres, a felhasználót belépteti a rendszerbe, és egy session-t hoz létre.
- **messages.success**: Tájékoztató üzenetek küldése a felhasználónak, sikeres vagy sikertelen bejelentkezés esetén.

4.2.3 Elfelejtett jelszó

Az elfelejtett jelszó funkció olyan kulcsfontosságú elem, amely megkönnyíti a felhasználók számára a hozzáférés visszaállítását az alkalmazásukhoz, amennyiben elfelejtették a jelszavukat. Az alkalmazás lehetővé teszi, hogy a felhasználók e-mail címük megadásával jelszó-visszaállítási kérelmet indítsanak. Az erre a célra kialakított felület intuitív módon vezeti végig a felhasználókat a folyamaton, különböző nézeteket biztosítva az egyes lépésekhez, például az e-mail megadásától kezdve egészen a jelszó sikeres módosításáig. Az implementáció során a Django beépített PasswordResetView-ra és kapcsolódó nézeteire támaszkodok, amelyeket testreszabott sablonokkal alakítottam a projekt specifikus igényeihez.

Fontos kiemelni, hogy bár a Django rendelkezik alapértelmezett jelszó-visszaállítási sablonokkal, a projektben saját sablonok lettek létrehozva, hogy a felület jobban illeszkedjen a designhoz és a funkcionalitáshoz. Ezek a sablonok az `urls.py`-ban lettek megadva az egyes útvonalakhoz, például a jelszó-visszaállítás kezdeményezéséhez (`password_reset/`), az értesítés megjelenítéséhez (`password_reset/done/`), a jelszó módosítás megerősítéséhez (`reset/<uidb64>/<token>/`), valamint a folyamat lezárásához (`reset/done/`). A sablonok a 13. ábrán láthatóak.



13. Ábra: Elfelejtett jelszó funkció oldalai

4.2.3.1 Beállítások és konfigurációk

A funkcióhoz szükséges e-mail küldés az alkalmazás settings.py fájlban került konfigurálásra. Az érzékeny adatok, mint az e-mail cím és jelszó, egy külön fájlban (email_password.py) tárolódnak, amelyek nem kerülnek tárolásra a távoli verziókezelőrendszerben.

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = host_user
EMAIL_HOST_PASSWORD = host_password
DEFAULT_FROM_EMAIL = EMAIL_HOST_USER
```

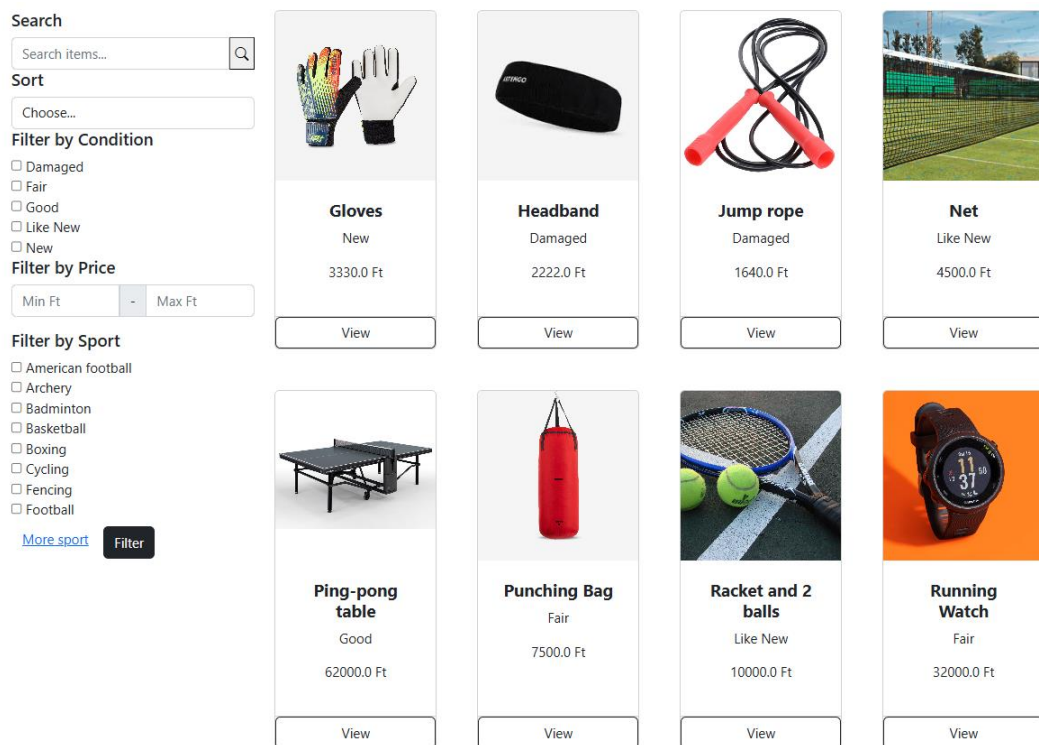
4.2.4 Kijelentkezés

A kijelentkezés funkció alapvető része egy webalkalmazás felhasználói autentikációs rendszerének, mivel lehetőséget biztosít a felhasználók számára, hogy biztonságosan kiléphessenek az aktív munkamenetükből. A kijelentkezési funkció megvalósítása során törekedtem arra, hogy a felhasználói élmény zökkenőmentes és egyszerű legyen, így nem volt szükség külön logout.html sablon létrehozására. Ehelyett a rendszer az események feldolgozását követően azonnal visszairányítja a felhasználót a kezdőlapra, miközben egy értesítés tájékoztatja őt a sikeres kijelentkezésről.

4.3 Home page

4.3.1 Általános áttekintés

A Home Page az alkalmazás központi eleme, ahol a felhasználók böngészhetnek a nem eladott sporteszközök kínálatát, szűrhetik azokat különböző paraméterek alapján, és részletesebb információkat szerezhetnek egy-egy termékről. Az eladott termékek nem jelennek meg a Home page-en, így az eladott termékkel kapcsolatban nem lehet kapcsolatba lépni a hirdetővel. Továbbá, ha a felhasználó be van jelentkezve, akkor a Home page-en nem jelennek meg a hirdetései, hogy saját magával ne tudjon kapcsolatba lépni a hirdető. A 14. ábrán látható az oldal, melynek célja, hogy a felhasználók számára átlátható és kényelmes keresési lehetőséget biztosítson az elérhető eszközök között.



14. Ábra: Home page felülete

Az oldal az alábbi fő funkciókat nyújtja:

- **Keresés és szűrés:** A felhasználók kulcsszavak megadásával vagy különböző szűrési opciók használatával (sportág, állapot, ár) szűkíthetik le a találatokat.
- **Rendezés:** A találatokat név vagy ár szerint növekvő vagy csökkenő sorrendbe állíthatják.

- **Kártyanézetű terméklista:** Az eszközök egy kártyarendszerben jelennek meg, amely tartalmazza a termék nevét, árát, állapotát és egy képet.
- **Lapozás:** A találatok lapozható formában jelennek meg, hogy a nagy mennyiségű tartalom kezelhető legyen.
- **Dinamizmus:** Az oldal tartalmaz dinamikus elemeket, például további sportágak listáját, amelyek megjelenítése a felhasználói interakcióktól függ.

4.3.2 Home page megvalósítása

A **Home Page** fejlesztése során a Django keretrendszer szolgáltatásaira és a StartBootstrap Shop Homepage[35] sablonra támaszkodtam a vizuális elemek és funkcionalitás megvalósításához. Az oldal szoros integrációt valósít meg a backenddel, amely az adatok dinamikus megjelenítését és a felhasználói interakciók kezelését teszi lehetővé.

A statikus fájlok és a médiafájlok kezelésére a **Django** keretrendszerben a következő beállítások kerültek megadásra a settings.py fájlban:

```
STATIC_URL = 'static/'
STATICFILES_DIRS = ['static/']
MEDIA_URL = 'media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

Ez lehetővé teszi a statikus tartalmak (CSS, JavaScript, képek) kezelését, valamint a felhasználók által feltöltött médiafájlok (például termékképek) megfelelő tárolását és elérését.

4.3.2.1 Backend logika: adatszűrés, rendezés

A home metódus kulcsfontosságú szerepet játszik a felhasználói lekérdezések kezelésében, az adatbázisból származó adatok hatékony szűrésével és rendezésével. A Django ORM segítségével a metódus dinamikusan állítja össze a lekérdezéseket a felhasználó által megadott szűrési feltételek alapján. **Főbb elemek részletesen:**

1. **filter:**

A filter metódus lehetővé teszi az adatbázisból származó objektumok szűrését adott feltételek alapján. Például:

- `name__icontains`: Ezzel az operátorral részleges szövegegyezés alapján kereshetünk az *Item* objektumok nevében.

- `price__gte` és `price__lte`: Az árintervallum szűrésére használható, ahol az alsó (`min_price`) és felső (`max_price`) határértékek alapján történik a lekérdezés.

2. **exclude:**

Az `exclude` metódus segítségével kizárhatók a találatok közül azok az objektumok, amelyek nem relevánsak a felhasználó számára. Például, a bejelentkezett felhasználók nem látják saját hirdetéseiket:

```
items.exclude(advertiser=request.user.profile)
```

3. **Dinamizmus:**

A szűrési feltételek teljesen dinamikusan épülnek fel a felhasználó által megadott paraméterek alapján, például:

- **Kulcsszó alapú keresés:** A `search_query` lekérdezési paraméter alapján.
- **Kategóriák vagy sportágak szűrése:** A `selected_sports` lista tartalma szerint.
- **Állapot szűrés:** A tárgyak állapotát meghatározó `selected_conditions` alapján.
- **Árintervallum:** A felhasználó által megadott minimum és maximum ár értékek alapján.

4. **Skálázhatóság:**

A Django ORM által biztosított optimalizációk miatt a szűrés nagy mennyiségű adat esetén is hatékonyan működik. Az `items` lekérdezéslánc minden egyes lépése újabb adatbázis-lekérdezést hoz létre, amely csak az adott lépésben releváns rekordokat tartalmazza.

A rendezés logikáját a `sort_option` változó tartalma határozza meg. Az `order_by` az adatok sorrendjének meghatározására szolgáló metódus. A `-` előtag csökkenő sorrendet eredményez.

4.3.2.2 Lapozás

A találatok kezelhetőségének érdekében a lapozás a Django **Paginator** osztályára épül:

```
paginator = Paginator(items, 8)
page_number = request.GET.get('page')
page_obj = paginator.get_page(page_number)
```

A Paginator a találatokat 8 elemre bontja oldalanként, amely javítja az oldal teljesítményét és a felhasználói élményt. A `get_page` a kívánt oldalszám alapján adja vissza az adott oldalon lévő elemeket, biztosítva a felhasználói navigáció egyszerűségét.

4.3.2.3 Frontend integráció

A backend által szolgáltatott adatok a **home.html** sablonban kerülnek megjelenítésre. A szűrők és rendezési lehetőségek interaktív HTML-elemek formájában jelennek meg, amelyek automatikusan továbbítják a paramétereket a backend felé.

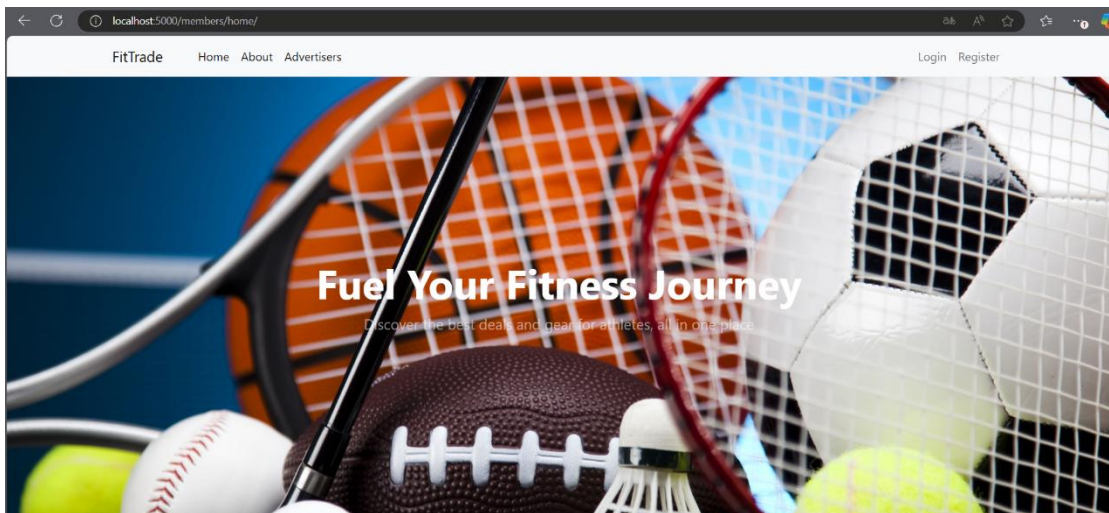
Például a szűrő űrlap egy része:

```
<form method="GET" action="{% url 'home' %}">
  <h5>Filter by Condition</h5>
  {% for condition in conditions %}
  <div>
    <input type="checkbox" name="conditions"
    value="{{condition.id }}"
    {% if condition.id|stringformat:"s" in
    selected_conditions %}checked{% endif %}>
    <label>{{ condition.name }}</label>
  </div>
  {% endfor %}
  <button type="submit" class="btn btn-dark mt-2">Filter</button>
</form>
```

- **{% for %}**: Django sablon-címke, amely a feltételek listáját iterálja.
- A szűrők az URL paraméterekkel együtt kerülnek továbbításra.
- **checked attribútum**: Dinamikusan hozzáadva, ha a feltétel már ki van választva.

4.4 Home, About és Advertisers oldalak tetején megjelenő képek

Az alkalmazás vizuális élményének javítása érdekében a **Home**, **About** és **Advertisers** oldalak tetején egy dinamikusan változó képsor jelenik meg. A 15. ábrán látható is, hogy éppen a Home page oldal tetején megjelenik a képsor egy képe.



15. Ábra: Home, About és Advertisers page tetején megjelenő képsor egy képe

4.4.1 Megvalósítás a backenden

A képek kezeléséért a `get_header_images()` függvény felelős, amely az alkalmazás **MEDIA_ROOT** mappájában található **header_images** alkönyvtárból tölti be az összes elérhető képet. Ezeket a képeket egy listába gyűjti, és URL-formátumban adja vissza, hogy a sablonok egyszerűen felhasználhassák őket. Ez a függvény dinamikusan tölti be a képeket, így bármikor hozzáadhatunk új képeket az **header_images** mappához anélkül, hogy további kódmódosítást kellene végezni.

```
def get_header_images():
    image_folder = os.path.join(settings.MEDIA_ROOT,
                                'header_images')
    image_urls = [
        f"{settings.MEDIA_URL}header_images/{image}" for
        image in os.listdir(image_folder)
    ]
    return image_urls
```

A **Home** és **About** oldalak megjelenítési logikája hasonló: a sablonmotor segítségével a nézetből kapott képlistát JSON-formátumban továbbítja az oldalhoz, hogy azt a frontend kezelhesse. Az **About** oldal például a következőképpen használja fel a képlistát:

```
return render(request, 'about.html', { 'images':  
    json.dumps(image_urls) })
```

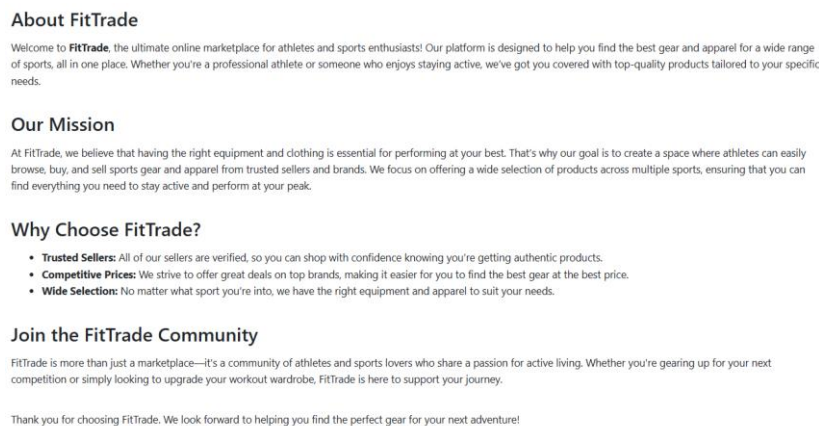
4.4.2 Megvalósítás a frontenden

A frontenden HTML és JavaScript-tel lett megvalósítva. A JavaScript kódból kiemelendők:

- **JSON.parse('{ images|safe })')**: Ez a metódus a Pythonból továbbított képlistát JSON-formátumból JavaScript objektummá alakítja át.
- **setInterval(changeBackground, 5000)**: Ezzel a függvénnyel automatikusan 5 másodpercenként váltja a fejléc háttérképét.
- **header.style.backgroundImage**: Ez a JavaScript utasítás határozza meg az aktuális háttérképet a fejlécben.

4.5 About page

Az About page célja, hogy a FitTrade platform bemutatásával egyértelműen és vonzó módon kommunikálja a weboldal küldetését, értékeit és céljait a látogatók számára. A 16. ábrán látható az az About page tartalma.

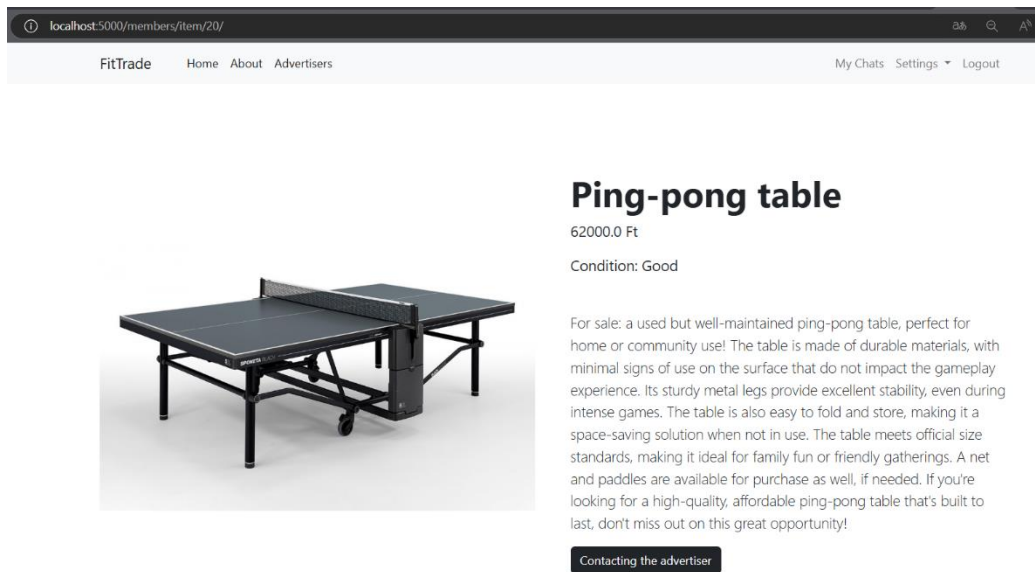


16. Ábra: About page

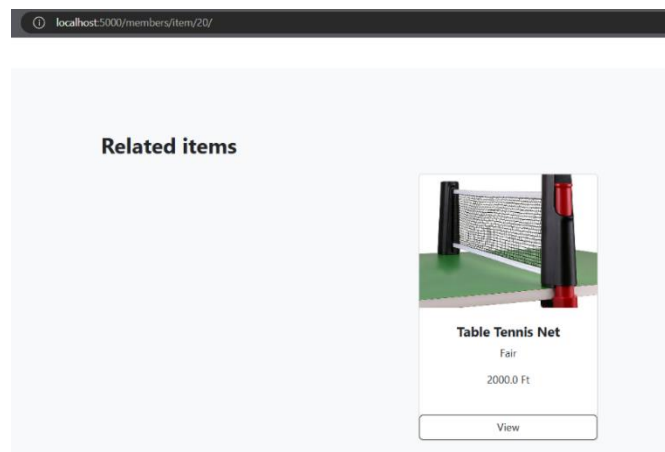
4.6 Hirdetés részletes oldala

A Hirdetés részletes oldalának célja, hogy a felhasználók számára egy adott termék részletes információit (termék neve, ára, állapota, leírása és képe) átlátható és vonzó formában jelenítse meg, miközben releváns kapcsolódó hirdetéseket (sportághoz kapcsolódó elérhető termékeket jelenít meg) is ajánl. Az oldal lehetőséget biztosít a regisztrált felhasználók számára, hogy kapcsolatba lépjenek a hirdetővel. Az

oldal kinézetének megvalósításakor felhasználtam a Start Bootstrap: Shop Item Template sablont [36]. Az oldalt szemlélteti a 17. és 18. ábra.



17. Ábra: Hirdetés részletes oldalának felső része



18. Ábra: Hirdetés részletes oldalának alsó része

4.6.1 Megvalósítás részletei

4.6.1.1 Útvonal definiálása

A következő útvonal definíció hozzáadása biztosítja, hogy az egyedi azonosítók alapján elérhetőek legyenek az egyes hirdetések részletes oldalai:

```
path('item/<int:pk>/', views.item, name='item'),
```

Ez a minta lehetővé teszi, hogy az URL tartalmazza a termék azonosítóját (pk), amely alapján a megjelenítendő hirdetés lekérdezhető.

4.6.1.2 Az item nézet

Az **item nézet** a Django keretrendszer egyik alapvető funkcióját használja a felhasználói kérések kiszolgálására és az adatok előkészítésére a megjelenítéshez. Ez a metódus biztosítja, hogy a kért hirdetés adatai biztonságosan és hatékonyan kerüljenek feldolgozásra, valamint releváns kapcsolódó hirdetéseket is biztosít a felhasználó számára.

Az **get_object_or_404** függvény a Django egyik kritikus eleme, amely biztosítja, hogy az adott hirdetés valóban létezik az adatbázisban. Amennyiben a hirdetés nem található, egy 404-es hibát dob, ezzel megakadályozva az alkalmazás további nem kívánt működését.

A nézet további fontos részét képezi a **kapcsolódó hirdetések szűrése**, amely során az aktuális hirdetés azonosítóját kizárják a listából. A szűrés két feltételen alapul: az adott termék még nem került eladásra (**sold=False**) és ugyanahhoz a sportkategóriához tartozik, mint az aktuális hirdetés (**sport=current_item.sport**). Ezek a szempontok együttesen releváns és kontextusba illő találatokat eredményeznek.

A **render függvény** által az előkészített adatok egy sablonban kerülnek megjelenítésre.

4.6.1.3 HTML struktúra és sablonlogika

A termék adatai dinamikusan, a megadott Django template változók (`{{ item.<mező> }}`) segítségével kerülnek megjelenítésre.

A kapcsolódó hirdetések egy dinamikusan generált szekcióban jelennek meg, amely egy iterációs sablonlogikát alkalmaz (`{% for related_item in related_items %}`). Ez a logika biztosítja, hogy az adatbázisból kinyert összes releváns találat egy egységes formátumban kerüljön megjelenítésre. Amennyiben nincsenek kapcsolódó hirdetések, az `{% empty %}` kulcsszóval definiált alternatív tartalom kerül megjelenítésre, amely egy egyszerű, de informatív üzenet a felhasználónak.

A sablon az aktuális felhasználó hitelesítési állapotának figyelembevételével biztosít különböző funkciókat. Az `{% if user.is_authenticated %}` feltétel segítségével a rendszer csak a bejelentkezett felhasználók számára teszi elérhetővé a

hirdetővel való kapcsolatfelvétel gombját. Az anonim felhasználók figyelmeztető üzenetet kapnak arról, hogy be kell jelentkezniük ahhoz, hogy elérjék ezt a funkciót.

4.7 Hirdetés kezelő (CRUD) oldalai

4.7.1 My Ads oldal

A "My Ads" felület célja, hogy a bejelentkezett felhasználók számára egy áttekinthető és könnyen kezelhető felületet biztosítson az általuk közzétett hirdetések kezelésére. Ez az oldal lehetőséget nyújt a hirdetések megtekintésére, szerkesztésére, új hirdetések hozzáadására, valamint több kijelölt hirdetés egyszerre történő törlésére.

Új hirdetés hozzáadására az „Add New” gomb szolgál a My Ads felületen, ugyanis egy új hirdetés létrehozására szolgáló oldalra irányítja a felhasználót. A My Ads oldalon az adott hirdetés nevére kattintva a hirdetés szerkesztő felületére irányítja a felhasználót. A felület megtekinthető a 19. ábrán.

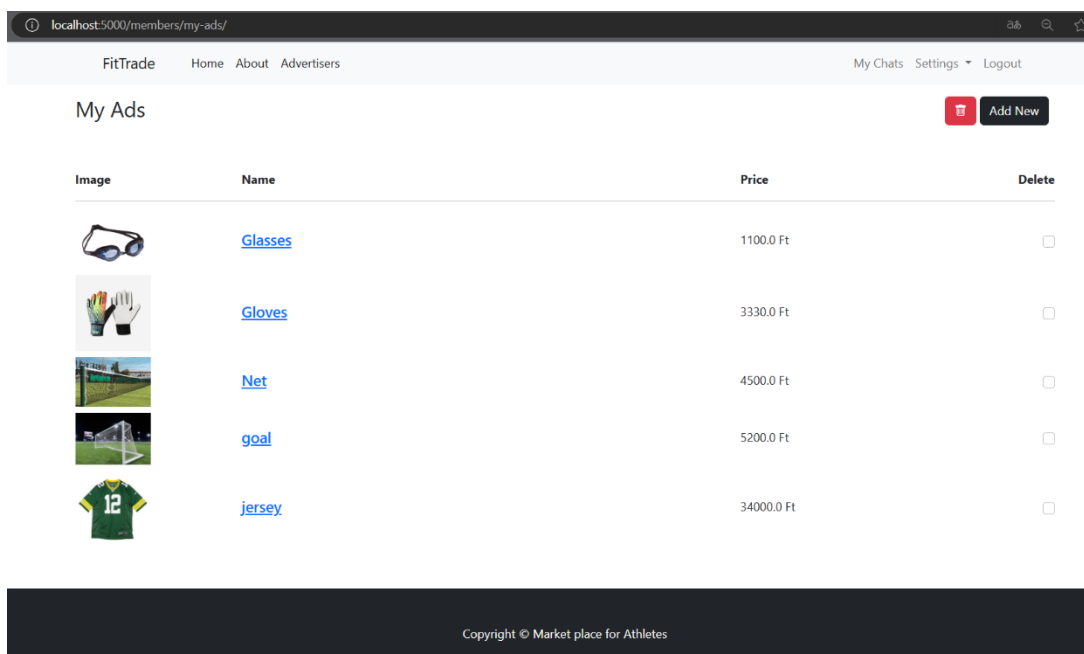







Image	Name	Price	Delete
	Glasses	1100.0 Ft	<input type="checkbox"/>
	Gloves	3330.0 Ft	<input type="checkbox"/>
	Net	4500.0 Ft	<input type="checkbox"/>
	goal	5200.0 Ft	<input type="checkbox"/>
	jersey	34000.0 Ft	<input type="checkbox"/>

Copyright © Market place for Athletes

19. Ábra: My Ads oldal

4.7.1.1 Megvalósítás részletei

A `@login_required` dekorátor biztosítja, hogy az oldal kizárólag bejelentkezett felhasználók számára érhető el. Ha egy nem hitelesített felhasználó próbálja meg elérni a funkciót, a rendszer automatikusan a bejelentkezési oldalra irányítja át. A bejelentkezett felhasználóhoz kapcsolódó hirdetések az `Item.objects.filter(advertiser=user_profile)` lekérdezéssel kerülnek előkészítésre. A POST-kérés logikája lehetővé teszi a felhasználó számára, hogy

egyszerre több hirdetést is töröljön. A rendszer a `request.POST.getlist('item_ids')` metódussal olvassa be a kijelölt elemek azonosítóit, majd a `Item.objects.filter(id__in=item_ids, advertiser=user_profile).delete()` utasítással törli azokat.

A sablon iterációs logikát alkalmaz a hirdetések megjelenítéséhez: a `{% for item in user_items %}` ciklus minden egyes hirdetéshez egy kártyát generál, amely tartalmazza az alapvető információkat és interakciós elemeket, például a szerkesztési linket és a törlési jelölőnégyzetet. A sablon egy speciális formot biztosít a hirdetések egyszerre történő törléséhez. A `<input type="checkbox" name="item_ids" value="{{ item.id }}"` elemek segítségével a felhasználók több hirdetést is kiválaszthatnak, amelyeket a sablon logikája POST-kéréssel továbbít a kiszolgálónak a törlés végrehajtására.

4.7.2 Új hirdetés létrehozása oldal

Az új hirdetés létrehozása oldal lehetővé teszi a bejelentkezett felhasználók (itt is használva van a `@login_required` dekorátor) számára, hogy sporteszközökről szóló hirdetéseket adhassanak hozzá az adatbázishoz. A művelet során a rendszer validálja a megadott adatokat, biztosítva ezzel a hirdetések konzisztenciáját és minőségét. A hirdetés létrehozása egy form alapú megoldáson keresztül történik, amely egyértelmű visszajelzéseket ad a felhasználónak sikeres vagy sikertelen feltöltés esetén. Az új hirdetés létrehozásának formja megtekinthető a 20. ábrán.

The screenshot displays the 'Item adding form' within the FitTrade application. The form is structured as follows:

- Item Name:** A text input field.
- Sport:** A text input field with a placeholder '.....'.
- Condition:** A text input field with a placeholder '.....'.
- Item Description:** A large text area for detailed description.
- Price (HUF):** A text input field.
- Upload Image:** A section containing a file selection button labeled 'Fájl kiválasztása' and a status indicator 'Nem lett kiválasztva fájl'.
- Submit:** A button to submit the form.

The footer of the page indicates 'Copyright © Market place for Athletes'.

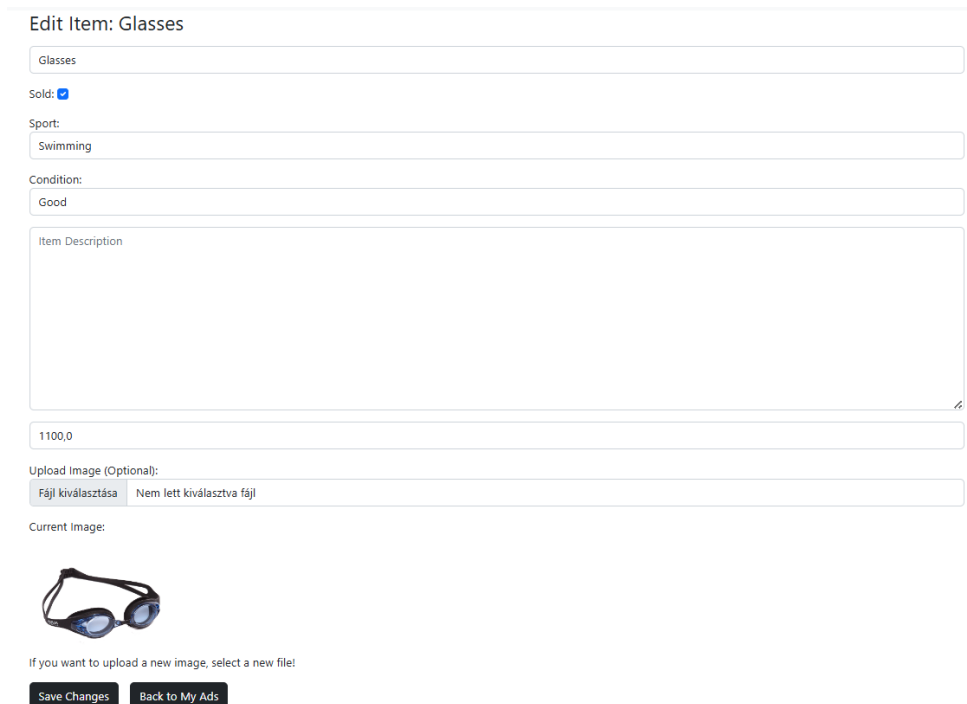
20. Ábra: Új hirdetés létrehozó oldala

A `CreateListingForm` osztály biztosítja az űrlap mezőinek validációját és a felhasználói élmény javításához szükséges attribútumokat. A mezők, mint például a `name`, `price`, `description` és `image`, a `forms.ModelForm` metódusait használják a könnyű testreszabhatóság érdekében. Az árfeldolgozás részeként a `clean_price` metódus gondoskodik arról, hogy negatív értékek és nulla ne kerülhessenek mentésre. Az űrlapot a `createlisting` nézet kezeli, amely a POST kéréseket dolgozza fel és menti a hirdetést, ha az űrlap érvényes. A messages rendszer segítségével a felhasználók közvetlen visszajelzést kapnak.

4.7.3 Hirdetés szerkesztő felülete

A hirdetések szerkesztése lehetővé teszi a bejelentkezett felhasználók (itt is használva van a `@login_required` dekorátor) számára, hogy a meglévő hirdetéseiket frissítsék, például módosítsák a termék eladási státuszát, annak következtében hogy már elkelt. A szerkesztési folyamat különös figyelmet fordít az adatok előzetes betöltésére, így a felhasználók könnyen és gyorsan végezhetik el a kívánt változtatásokat.

Itt az `EditListingForm` osztály biztosítja az űrlap mezőinek validációját. Továbbá itt is biztosítva van, hogy negatív értékek és nulla ne kerülhessenek mentésre az árnál. A szerkesztő oldal felülete megtekinthető a 21. ábrán.



The screenshot shows a web form titled "Edit Item: Glasses". It contains several input fields and a text area:

- A text input field with the value "Glasses".
- A "Sold:" checkbox that is checked.
- A "Sport:" dropdown menu with "Swimming" selected.
- A "Condition:" dropdown menu with "Good" selected.
- A large text area labeled "Item Description" which is currently empty.
- A price input field with the value "1100,0".
- An "Upload Image (Optional):" section with two buttons: "Fájl kiválasztása" (selected) and "Nem lett kiválasztva fájl".
- A "Current Image:" section showing a small image of a pair of swimming goggles.
- At the bottom, a message says "If you want to upload a new image, select a new file!" followed by two buttons: "Save Changes" and "Back to My Ads".

21. Ábra: Hirdetés szerkesztő oldala

4.8 My Profile oldal

Az **"My Profile"** felület célja, hogy lehetőséget biztosítson a felhasználóknak személyes adataik, például nevük, email címük és egyéb profilhoz kapcsolódó információik frissítésére. Ez a felület egyaránt lehetőséget nyújt a felhasználói és profilinformációk kezelésére egyetlen jól szervezett formában. Az adatváltoztatások után a rendszer azonnal frissíti a változtatásokat, és értesítést küld a sikeres módosításról vagy hibáról.

A megvalósításban két különálló form került felhasználásra: a `UserEditForm` a felhasználói adatok frissítésére, míg a `ProfileSignUpForm` a profiladatok módosítására. A kulcsfontosságú elemek közé tartozik a `request.user`-hez rendelt adatok frissítése, valamint az űrlap validációja és mentése, amelyek biztosítják a felhasználói élmény zökkenőmentességét. A nézet egyszerre két űrlapot renderel, amelyek közvetlenül HTML formában jelennek meg a felhasználó számára. A My Profile oldal felülete látható a 22. ábrán.


The screenshot shows a web browser at the URL `localhost:5000/members/my_profile/`. The page layout includes a top navigation bar with 'FitTrade', 'Home', 'About', and 'Advertisers' on the left, and 'My Chats', 'Settings', and 'Logout' on the right. The main content area is titled 'My Profile' and is split into two sections. The 'User Info' section on the left has four input fields: the first contains 'matyi' with a note 'Required. 150 characters or fewer. Letters, digits and @/+/./_ only.', the second contains 'Matyas', the third contains 'Sebes', and the fourth contains 'sebes@gmail.com'. Below these is a 'Change Password' button. The 'Profile Info' section on the right has a 'Dateofbirth' field with a calendar icon and a text area containing the text 'I'm a Hungarian guy.'. At the bottom of the 'User Info' section is a 'Save Changes' button. The footer is a dark bar with the text 'Copyright © Market place for Athletes'.

22. Ábra: My Profile felülete

4.8.1 Jelszó változtatás

Ez a funkció lehetővé teszi a felhasználók számára, hogy megváltoztassák jelszavukat. Ez a felület megtekinthető a 23. ábrán. A felhasználók először megadják a régi jelszavukat, majd kétszer az új jelszavukat a változtatás érvényesítéséhez. Az új jelszó mentése után a rendszer automatikusan frissíti az aktív munkamenetet, hogy elkerülje a felhasználó kijelentkezését.

A kulcsfontosságú a `UserPasswordChangeForm`, amely az űrlap validációját végzi, valamint az `update_session_auth_hash` funkció, amely a felhasználó munkamenetének megőrzéséért felelős. A validáció hibái esetén a rendszer a megfelelő üzeneteket jeleníti meg, hogy a felhasználók könnyen korrigálhassák a beviteli hibákat.



23. Ábra: Jelszó változtatás felülete

4.9 Chat

4.9.1 WebSocket alapú chat elvei

A WebSocket technológia olyan kommunikációs protokollt biztosít, amely állandó, kétirányú kapcsolatot tart fenn a kliens és a szerver között. Ez különösen hasznos valós idejű alkalmazások, például chat rendszerek esetén, mivel az üzenetek azonnal továbbíthatók mindkét irányba, megszüntetve a hagyományos HTTP-kérések szükségességét.

A WebSocket egy tartós kapcsolatot hoz létre, amely során a kliens és a szerver közvetlenül kommunikálhat, jelentősen csökkentve a késleltetést és az erőforrásigényt. A Django Channels keretrendszer a hagyományos Django környezetre épülve ad hozzá WebSocket-támogatást, és lehetőséget biztosít a skálázható, valós idejű alkalmazások fejlesztésére.

4.9.2 Konfiguráció és beállítások

4.9.2.1 Daphne telepítése

A Daphne egy aszinkron szerver, amely az ASGI-protokoll kezelésére szolgál, és elengedhetetlen a Django Channels által kínált valós idejű kommunikációs funkciók működtetéséhez.

4.9.2.2 ASGI és WSGI beállítások

Az alábbi beállításokkal az ASGI-t állítottam be, amely lehetővé teszi az aszinkron kommunikációt a Django alkalmazásban:

```
WSGI_APPLICATION = 'MarketPlaceForAthletes.wsgi.application'
ASGI_APPLICATION = 'MarketPlaceForAthletes.asgi.application'
```

A WSGI továbbra is biztosítja a hagyományos HTTP-alapú működést, míg az ASGI szükséges a WebSocket kapcsolatokhoz és más aszinkron folyamatokhoz.

4.9.2.3 Channel Layers konfiguráció

A Channel Layers a WebSocket alapú üzenetek továbbításáért felelős. Az alábbi kód az InMemoryChannelLayer használatával állítja be a Channel Layer-t, amely a fejlesztés korai szakaszában ideális választás, mivel nem igényel külső adatbázist:

```
CHANNEL_LAYERS = {
    "default": {
        "BACKEND": "channels.layers.InMemoryChannelLayer"
    },
}
```

A fejlesztés későbbi szakaszaiban, nagyobb forgalom kezelésére ajánlott a Redis vagy más tartós megoldásra váltani.

4.9.2.4 ASGI konfiguráció módosítása

Az alábbi konfiguráció biztosítja, hogy az alkalmazás képes legyen kezelni a HTTP kéréseket és a WebSocket kapcsolatokat, míg az autentikációs köztes réteg (AuthMiddlewareStack) lehetővé teszi, hogy az azonosított felhasználók kizárólagos hozzáférést kapjanak a valós idejű chathez.

```
application = ProtocolTypeRouter(
    {
        "http": get_asgi_application(),
        "websocket": AllowedHostsOriginValidator(
            AuthMiddlewareStack(URLRouter(chat.routing.websocket_urlpatterns))
        ),
    })
```

4.9.3 Consumer osztályok

A consumers.py fájl központi szerepet játszik az aszinkron kommunikáció logikájában, mivel itt kerülnek implementálásra a WebSocket-ek kezeléséért felelős **Consumer osztályok**. Ezek az osztályok biztosítják, hogy a WebSocket alapú

kommunikáció mindkét irányba – szerverről kliensre és fordítva – zökkenőmentesen működjön.

4.9.3.1 Kapcsolódás kezelése

A `connect` metódus automatikusan lefut, amikor egy kliens megpróbál csatlakozni a WebSocket-en keresztül. Itt azonosítja a felhasználót és hozzárendeli őt egy adott "szobához" vagy csoporthoz, amelynek az azonosítója a felhasználó egyedi azonosítója. A `group_add` metódus segítségével a WebSocket kapcsolatot beilleszti a megfelelő csoportba, amely lehetővé teszi az üzenetek hatékony továbbítását.

4.9.3.2 Kapcsolat bontása

A `disconnect` metódus felelős a kliens WebSocket kapcsolatának megszakításáért, és gondoskodik arról, hogy a kapcsolat a megfelelő csoportból is eltávolításra kerüljön. Ez elengedhetetlen a rendszer memóriakezelésének optimalizálásához.

4.9.3.3 Üzenet fogadása

A `receive` metódus a kliens által küldött üzenetek feldolgozását végzi. A JSON formátumban érkező üzeneteket elemezi, és a tartalmuk alapján hoz létre új chat vagy üzenet objektumokat. Az üzenetek feldolgozása során fontos, hogy a felhasználók (küldő és fogadó), valamint az esetleges kapcsolódó tárgyak azonosítva legyenek, és az üzenet tartalma a megfelelő adatbázis-modellben rögzüljön.

4.9.3.4 Események kezelése

Az eseményvezérelt rendszer biztosítja, hogy az újonnan érkezett üzenetek azonnal továbbításra kerüljenek a megfelelő kliensekhez. Az események kezeléséért felelős metódusok, például a `chat_message`, biztosítják, hogy a kliens a szerver által generált üzeneteket valós időben megkapja.

4.9.3.5 Jelzések és valós idejű értesítések

A fájl tartalmaz egy `@receiver` dekorátorral ellátott jelzési rendszert, amely a Django modelljeihez kapcsolódó eseményeket kezeli. Amikor például egy új üzenet jön létre, a Django által támogatott `post_save` jelzés alapján valós idejű értesítéseket küld a megfelelő WebSocket csoportoknak, biztosítva ezzel a valós idejű kommunikációt a felhasználók között.

4.9.4 Routing

A `routing.py` fájl szerepe a WebSocket alapú kommunikációs csatornák útvonalainak konfigurációja, amely az alkalmazás különböző szegmensei közötti adatáramlást biztosítja. A WebSocket alapú URL-ek általában `ws/` előtaggal vannak definiálva, jelezve a WebSocket protokoll használatát.

```
from django.urls import path
from . import consumers
websocket_urlpatterns = [
    path("ws/chat/", consumers.ChatConsumer.as_asgi()),
]
```

4.9.5 Chat backend logikája

A Chat nézetei, amelyek az üzenetküldési funkciók logikáját és az adatbázis műveletek kezelését biztosítják. Az alkalmazás alapvető funkcionalitását meghatározó nézetek, például a `chat` és a `getChats`, a felhasználók közötti kommunikáció megvalósítását támogatják, azzal a céllal, hogy az aktuális felhasználóhoz kapcsolódó üzeneteket és csevegéseket hatékonyan kezelje.

A `chat` nézet a felhasználó hitelesítésének ellenőrzésére helyez hangsúlyt, biztosítva, hogy csak bejelentkezett felhasználók férhessenek hozzá az üzenetküldési funkciókhoz, miközben dinamikusan rendereli a csevegőszobát az aktuális felhasználó azonosítójának figyelembevételével.

A `getChats` nézet kulcsfontosságú szerepet játszik a felhasználók közötti üzenetfolyamok előállításában és strukturált adatként való visszaküldésében, lehetővé téve a frontend számára az aktív csevegések és az ezekhez tartozó üzenetek elérését. Ez a nézet a csevegések részletes adatainak előállításához több különböző adatbázismodellt (például **Chat** és **Item**) használ, majd JSON formátumban küldi vissza azokat a kliens számára. A lekérdezések során például az adott hirdetéshez kapcsolódó új csevegések automatikus létrehozására is sor kerülhet, amennyiben az még nem létezik, és az utolsó üzenet tartalmát is beágyazza a válaszbá.

A `getConversation` nézet célja a konkrét csevegési folyamat üzeneteinek előhívása és azok feldolgozása, beleértve az olvasási állapot frissítését is, hogy a felhasználó visszajelzést kapjon az üzeneteiről. Az üzenetek részletei – például küldő

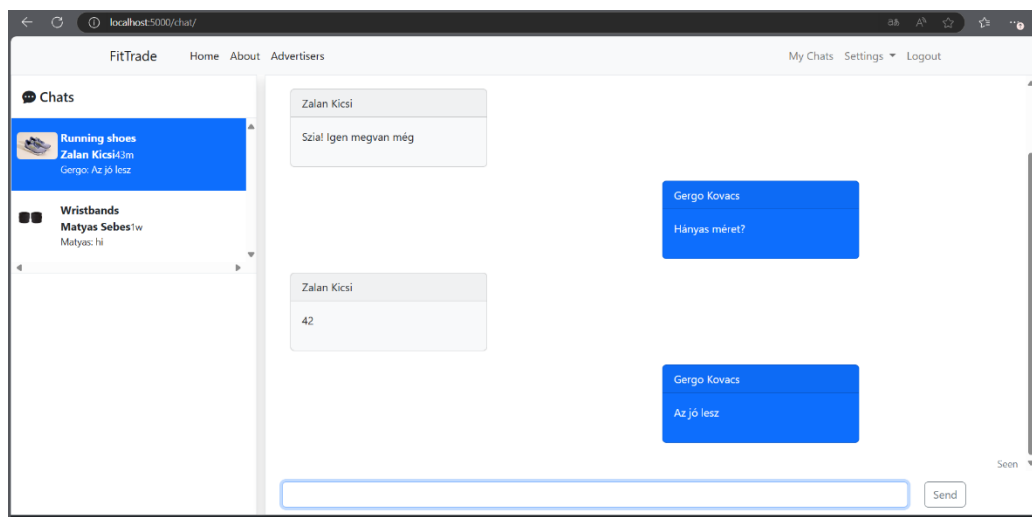
és fogadó adatok, időbélyeg és olvasottsági állapot – szintén JSON formátumban kerülnek visszaküldésre, amely a felhasználói élmény zavartalanságát szolgálja.

4.9.6 Chat frontend logikája, My Chats oldal

A **room.html** a frontend megjelenítés alapját képező HTML sablon, amely a valós idejű csevegés vizuális és funkcionális megvalósításáért felelős, miközben a Bootstrap és JavaScript dinamikus elemeivel biztosítja a modern és rezponzív kialakítást. A sablon két fő részből áll: az oldalsávból, amely a felhasználók és csevegések listáját tartalmazza, és a fő csevegési területből, ahol az üzenetváltások zajlanak.

Az oldalsáv az aktív csevegések áttekinthetőségét szolgálja, ahol a listában megjelenő elemek tartalmazzák a hirdetések nevét, a beszélgetőpartner felhasználónevét, utolsó üzenet kivonatát valamint, hogy az utolsó üzenetváltás körülbelül mennyi ideje történt, ezek az adatok segítenek a felhasználónak azonnal azonosítani az adott csevegést. A JavaScript kód felelős az oldalsáv dinamikus frissítéséért, amely a szerverről kapott adatokat AJAX kéréseken keresztül dolgozza fel, miközben biztosítja, hogy a felhasználók gyorsan válthassanak az egyes beszélgetések között. Az oldalsávban lévő chat-ek aktualitás szerint vannak rendezve, tehát mindig a legfrissebb beszélgetés van az oldalsáv tetején, legrégebbi az oldalsáv alján.

A fő csevegési terület, amely a valós idejű üzenetküldést valósítja meg, egy görgethető panelen keresztül jeleníti meg az üzeneteket, amelyek színkódokkal különülnek el a feladó kilététől függően. A fő csevegési területen mindig az chat jelenik meg amit a felhasználó kiválasztott az oldalsávban. Az üzenetek elküldése egy egyszerű szövegmező és gomb használatával történik, míg a valós idejű frissítéseket WebSocket kapcsolatok kezelik, biztosítva a gyors és folyamatos kommunikációt a kliens és a szerver között. Az adott üzenet olvasottsági állapotához egy vizuális jelölő is hozzájárul, ugyanis ha az adott üzenet olvasott, akkor egy halvány „Seen” felirat megjelenik a csevegési felületen. A 24. ábrán látható a My Chats felülete, míg a 25. ábrán megtekinthető a beszélgetésben elküldött JSON formátumú üzenetek felépítése.



24. Ábra: My Chats felület

```
Object {
  isread: true
  message: "42"
  receiver: {
    name: "Gergo Kovacs"
    pk: 9
  }
  sender: {pk: 11, name: 'Zalan Kicsi'}
  timestamp: "2024-10-30 10:56:36.632091+00:00"
}
```

```
Object {
  isread: true
  message: "Az jó lesz"
  receiver: {
    name: "Zalan Kicsi"
    pk: 11
  }
  sender: {
    name: "Gergo Kovacs"
    pk: 9
  }
  timestamp: "2024-11-25 08:39:05.263314+00:00"
}
```

25. Ábra: JSON formátumú üzenetváltás

4.10 Advertisers page

Az Advertisers oldal egy átfogó platform, amely a regisztrált hirdetőik adatainak – például nevük, életrajzuk és átlagos pontszámuk – megjelenítése mellett keresési és rendezési funkciókat kínál, hogy a látogatók könnyen megtalálják a számukra releváns hirdetők. Az oldalt a 26. ábra szemlélteti. Az értékelési rendszer a ScoreForm-ra épül, amely a ModelForm használatával validált 1-től 5-ig terjedő pontszámokat fogad el, garantálva az adatok helyességét az űrlap kitöltése során.

Az advertisers nézet GET paraméterek alapján végzi a hirdetők listájának szűrését és rendezését, az annotate metódussal pedig kiszámítja az átlagos pontszámokat, miközben a Paginator segítségével biztosítja a lapozott megjelenítést. Az add_score nézet az értékelések hozzáadását és frissítését kezeli, az update_or_create metódussal pedig egyszerre kezeli az új és meglévő értékeléseket, miközben csak bejelentkezett felhasználók számára érhető el. A HTML és CSS megvalósítás egy átlátható, accordion

alapú struktúrát biztosít, ahol a hirdetőik hirdetéseit görgethető galériában jelennek meg, az értékelési űrlapok pedig dinamikusan igazodnak a felhasználói tevékenységhez.

The screenshot shows a web interface for managing advertisers. On the left, there is a search bar labeled 'Search' with the placeholder text 'Search advertisers...' and a magnifying glass icon. Below it is a 'Sort' dropdown menu currently set to 'Score (High to Low)'. The main content area displays the profile of an advertiser named 'peti nagy' with an 'Average Score' of 4.5. Below the name is a 'Bio' section stating 'I am a 27-year-old Hungarian guy. In my free time I play football and swim.' Underneath the bio is an 'Advertisements' section featuring a carousel of images: a football and a pair of blue sneakers. Below the carousel is a button labeled 'Add New Score'. At the bottom, there is a list of other advertisers with their names and average scores, each with a dropdown arrow to its right:

Advertiser Name	Average Score
lajos balatoni	4.0
h h	4.0
nagy nagy	3.0
rek kissbb	2.5
dani pap	2.0
Harry Potter	2.0

26. Ábra: Advertisers felület

4.11 Adminisztrátor felülete

A rendszer adminisztrátorai számára a Django beépített adminisztrációs felületét alkalmaztam, amely egyszerű, de hatékony eszközt biztosít az adatok kezeléséhez. A felület látható a 27. ábrán. A fejlesztés során a projektben definiált összes entitásmodell regisztrálásra került a Django adminisztrációs felületére az `admin.site.register()` függvény segítségével, ez lehetővé teszi entitások teljes körű kezelését (CRUD).

The screenshot shows the Django administration interface. At the top, it says 'Django administration'. Below that is a section titled 'Site administration'. Under this section, there are several categories of models, each with a list of models and their corresponding actions (Add and Change):

Category	Model	Add	Change
AUTHENTICATION AND AUTHORIZATION	Groups	+ Add	Change
	Users	+ Add	Change
CHAT	Chat messages	+ Add	Change
	Chats	+ Add	Change
ITEMS	Item conditions	+ Add	Change
	Items	+ Add	Change
	Sports	+ Add	Change
MEMBERS	Profiles	+ Add	Change
	Scores	+ Add	Change

27. Ábra: Advertisers felület

5 Tesztelés

A projekt átfogó tesztelése érdekében a Django beépített Unit tesztjei mellett a fejlett pytest és coverage eszközöket is alkalmaztam, amelyek kiváló lehetőséget biztosítanak a kód minőségének és tesztek átfogó lefedettségének ellenőrzésére.

5.1 Pytest

A pytest a Python alapú projektek tesztelésére szolgáló eszközök egyik legismertebbike, amelynek használata a tesztek írását és futtatását egyaránt egyszerűvé és hatékonyá teszi. Django-alapú rendszerekben különösen hasznos, mivel zökkenőmentesen integrálódik a meglévő Unit tesztekhez, és számos kiegészítő funkciót kínál, mint például paraméterezett tesztelés és dinamikus adatelemzés [37].

5.2 Coverage

A coverage.py eszköz biztosítja, hogy pontosan mérni lehessen, mely kódrészeket fedik le a tesztek, ezáltal segítve a rejtett hiányosságok azonosítását és a kód minőségének javítását. A coverage részletes jelentéseket generál, amelyek nemcsak az általános lefedettségi százalékot, hanem a nem tesztelt kódsorok pontos helyét is megmutatják. További lehetőségként HTML alapú jelentéseket is készíthetünk, amelyek interaktív módon vizsgálhatók meg egy böngészőben: coverage `html`. Ezáltal a fejlesztők azonnal azonosíthatják azokat a kódrészeket, amelyeket a tesztek nem érintenek, lehetőséget teremtve a célzott javításokra és tesztbővítésekre [38].

5.3 Tesztelés folyamata

Az alkalmazás tesztelése során kiemelt figyelmet fordítottam arra, hogy a rendszer minden komponense, beleértve a modellekben definiált adatstruktúrákat és a nézetekben megvalósított funkcionalitást, alaposan és átfogóan legyen tesztelve. A tesztek a projekt mindhárom alkalmazásához – members, items, és chat – tartozó tests.py fájlokban valósítottam meg, biztosítva, hogy az egyes funkciók és adatkapcsolatok megfelelően működjenek. Összesen 34 teszt került kidolgozásra. A következő alfejezet bemutat 4 tesztet, hogy átfogó képet adjon a tesztelési stratégiáról és a kódban alkalmazott megoldásokról.

5.3.1 Bemutatott tesztesetek

5.3.1.1 Felhasználók kijelentkezésének tesztelése

Ez a teszt az `LogoutUserViewTests` osztály része, amely azt ellenőrzi, hogy a felhasználók sikeresen kijelentkeznek-e a rendszerből, valamint a kijelentkezési folyamat után megfelelően visszairányításra kerülnek-e a főoldalra.

A teszt előkészítési szakaszában a `setUp` metódus létrehoz egy tesztfelhasználót a `User.objects.create_user` segítségével, majd a Django kliens (`Client()`) használatával bejelentkezteti azt. Ez a folyamat lehetővé teszi a tesztek során a felhasználói interakciók szimulálását.

A `test_logout_user` metódusban a `self.client.get(reverse('logout'))` hívás biztosítja a kijelentkezési folyamatot, ahol a `reverse` kulcsszóval az URL-ek dinamikus generálása történik. Az eredményt a `self.assertRedirects` metódus segítségével ellenőrizzük, amely garantálja, hogy a válasz átirányítja a felhasználót a főoldalra (`reverse('home')`). Továbbá a `self.assertNotIn('_auth_user_id', self.client.session)` kifejezés igazolja, hogy a kijelentkezés után a felhasználó munkamenetéhez kapcsolódó információk sikeresen eltávolításra kerültek.

5.3.1.2 Új hirdetés létrehozásának tesztelése

Ez a teszt a `CreateListingViewTests` osztályban található, és a hirdetés létrehozásának folyamatát vizsgálja, amely az alkalmazás egyik kulcsfunkciója.

Az előkészítési szakaszban a `setUp` metódus hasonlóan működik, mint az előző tesztben, azonban itt további objektumok is létrejönnek: egy `Profile`, egy `Sport`, és egy `ItemCondition`, amelyek az új hirdetéshez szükségesek. A tesztfelhasználó szintén bejelentkezik a Django kliens segítségével.

A `test_create_listing_success` metódus egy `POST` kérés segítségével (`self.client.post(reverse('addlisting'), {...})`) szimulálja az új hirdetés létrehozását. Az itt szereplő `reverse` kulcsszó biztosítja, hogy a megfelelő URL-re történjen a kérés. Az adatok közé tartozik a hirdetés neve, ára, sportkategóriája és leírása, amelyek az alkalmazás adatbázisába kerülnek.

Az eredmények ellenőrzése során a `self.assertEqual(response.status_code, 302)` megerősíti, hogy a kérés sikeres volt, és a felhasználó

átírányításra került. A státusz kódnak a 302 lett megadva, mert ha sikeres volt az új termék felvitele, akkor átírányít az oldal. A `self.assertTrue(Item.objects.filter(name="Tennis Racket").first() != None)` pedig ellenőrzí, hogy az adatbázisban valóban létrejött az új hirdetés.

5.3.1.3 Üzenetek helyes kezelése teszt

Ez a teszt a `ChatModelTests` osztályban található, és a `Chat`, valamint a `ChatMessage` modellek működésének helyességét ellenőrzí, különösen azt, hogy a rendszer képes-e a résztvevő felhasználók és a kapcsolódó hirdetés alapján a megfelelő üzeneteket lekérnı. A teszt célja annak biztosítása, hogy az üzenetek mentése és lekérdezése összhangban legyen az elvárt viselkedéssel.

A `setUp` metódus gondoskodik a szükséges tesztkörnyezet létrehozásáról, ahol két felhasználó kerül regisztrálásra a `User.objects.create_user` segítségével, továbbá ezekhez profilokat is rendel. Ezenkívíl egy sportkategória, egy állapot, valamint egy hirdetés is létrejön, amelyhez a chat kapcsolódik.

A `test_get_chat_messages` metódus a teszt magját képezi. A chat létrehozása a `Chat.GetChatByParticipants(self.user1, self.user2, self.item)` metódussal történik, amely biztosítja, hogy a chat objektum az adott résztvevők és hirdetés alapján jöjjön létre. Az üzeneteket a `ChatMessage.objects.create` metódus segítségével adja hozzá, ahol megadja az üzenet küldőjét (`sender`), címzettjét (`receiver`), és tartalmát (`content`). A chathez kapcsolódó összes üzenetet a `chat.GetMessages()` metódussal kérdeztem le.

Az eredmények ellenőrzése során a `self.assertEqual(messages.count(), 2)` megerősíti, hogy pontosan két üzenet került mentésre, míg a `self.assertEqual(messages[0], message1)` igazolja, hogy az elsőként lekérdezett üzenet megegyezik az elsőként létrehozottal.

5.3.1.4 A `getConversation` nézet működésének tesztelése

Ez a teszt a `ChatViewTests` osztályban található, és a `getConversation` nézet funkcionalitását vizsgálja, amely egy adott chathez kapcsolódó üzeneteket JSON formátumban küldi vissza.

A `setUp` metódus biztosítja a szükséges tesztkörnyezetet. Emellett itt a felhasználó bejelentkezik a `self.client.login` metódus segítségével, hogy szimulálja az autentikált felhasználói állapotot.

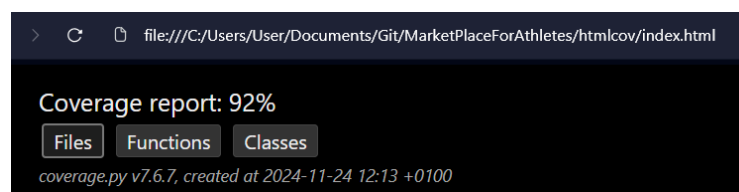
A `test_get_conversation` metódus egy GET kérés segítségével szimulálja a nézet hívását. A chat létrehozása a `Chat.GetChatByParticipants(self.user1, self.user2, self.item)` metódus használatával történik, majd a `self.client.get(f"/chat/get_conversation/?chat_id={chat.id}")` metódus egy GET HTTP kérést küld a nézetnek a megfelelő chat azonosítójával. A Django teszt kliens lehetővé teszi a HTTP kérések és válaszok szimulációját a tesztkörnyezetben.

Az eredmények ellenőrzése során a `self.assertEqual(response.status_code, 200)` igazolja, hogy a nézet sikeresen válaszolt, és a státuszkód 200-as értékkel tért vissza. A válasz JSON formátumba történő átalakítása a `response.json()` metódussal történik, amely egyszerű hozzáférést biztosít az adatokhoz. A `self.assertIn("messages", data)` állítás ellenőrzi, hogy a JSON válasz tartalmazza-e a "messages" kulcsot, amely az üzenetek listáját tárolja.

Ez a teszt igazolja, hogy a `getConversation` nézet megfelelően kezeli a bemenetként kapott chat-azonosítót, és helyesen küldi vissza a szükséges adatokat JSON formátumban. A `client.get` és a `response.json` használatával bemutatja, hogyan lehet a Django tesztkörnyezetben HTTP műveleteket szimulálni és az eredményeket ellenőrizni.

5.4 Lefedettség

A tesztkörnyezet alapos kidolgozottságát és a lefedettség hatékonyságát jól mutatja, hogy a `coverage html` eszköz elemzése alapján a kód **92%-os** lefedettséget ért el. Ez az érték egyértelműen jelzi, hogy az elkészült tesztek az alkalmazás szinte teljes funkcionalitását átfogják, miközben minimalizálják a nem tesztelt kódrészletek jelenlétét. A lefedettséget szemlélteti a 28. ábra.



28. Ábra: Lefedettség

6 Összefoglalás

A szakdolgozatban bemutatott Python Django-alapú market place alkalmazás egy ígéretes kiindulópontot jelent egy éles környezetben futó webalkalmazás létrehozásához. Az alkalmazás továbbfejlesztésével a felhasználók a jövőben bárhol, bármikor hozzáférhetnének a platformhoz, ami jelentősen növelné annak funkcionalitását és használati értékét.

A dolgozat elején részletesen ismertettem a fejlesztés során alkalmazott technológiákat, amelyek közül a legfrissebb verziókat használtam. Az éles üzembe helyezés során fontos változtatás lenne például az SQLite adatbázis lecserélése egy robusztusabb, skálázható megoldásra, amely jobban megfelel a valós felhasználói terhelésnek.

A tervezési folyamat részletes bemutatása során kitértem arra, hogy a market place megvalósítása az MVT modellt követi. Továbbá részletesen ismertettem az alkalmazáshoz szükséges modelleket.

A tervezés után bemutattam az alkalmazás megvalósítását a felhasználói felületen keresztül. Részleteztem, elmagyaráztam a market place egy-egy oldalának technikai megvalósítását.

Az utolsó fejezetben pedig ismertettem a market place tesztelését. Kitértem arra, hogy milyen eszközöket használtam a teszteléshez. Illetve kitértem még a tesztek által kapott lefedettségre is.

Irodalomjegyzék

- [1] Python Software Foundation: *Python 3 Documentation*, <https://docs.python.org/3/> (2024. nov. 28.)
- [2] Wikipedia: *Python (programming language)*, [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)) (2024. nov. 28.)
- [3] B. Simon, WebSockets, https://devil.iit.bme.hu/~sbalazs/onlab/websocket/SoiLesson06_WebSockets.pdf (2024. nov. 28.)
- [4] PEP 3333: *Python Web Server Gateway Interface v1.0.1*, <https://peps.python.org/pep-3333/> (2024. nov. 28.)
- [5] ASGI: *ASGI Documentation — ASGI 3.0 documentation*, <https://asgi.readthedocs.io/> (2024. nov. 28.)
- [6] Wikipedia: *Web framework*, https://en.wikipedia.org/wiki/Web_framework (2024. nov. 28.)
- [7] Wikipedia: *Django (web framework)*, [https://en.wikipedia.org/wiki/Django_\(web_framework\)](https://en.wikipedia.org/wiki/Django_(web_framework)) (2024. nov. 28.)
- [8] Django Software Foundation: *Django Documentation*, <https://docs.djangoproject.com/> (2024. nov. 28.)
- [9] Django Documentation: *Models*, <https://docs.djangoproject.com/en/stable/topics/db/models/> (2024. nov. 28.)
- [10] Django Documentation: *Templates*, <https://docs.djangoproject.com/en/stable/topics/templates/> (2024. nov. 28.)
- [11] Django Documentation: *Writing views*, <https://docs.djangoproject.com/en/stable/topics/http/views/> (2024. nov. 28.)
- [12] Rinu Gour: *Working Structure of Django MTV Architecture, Towards Data Science* (2024. nov. 28.)
- [13] Django Documentation: *The Django admin site*, <https://docs.djangoproject.com/en/stable/ref/contrib/admin/> (2024. nov. 28.)
- [14] Django Documentation: *User authentication in Django*, <https://docs.djangoproject.com/en/stable/topics/auth/> (2024. nov. 28.)

- [15] Django Documentation: *Making queries*, <https://docs.djangoproject.com/en/stable/topics/db/queries/> (2024. nov. 28.)
- [16] Django Documentation: *Many-to-one relationships*, https://docs.djangoproject.com/en/stable/topics/db/examples/many_to_one/ (2024. nov. 28.)
- [17] Django Documentation: *Database transactions*, <https://docs.djangoproject.com/en/stable/topics/db/transactions/> (2024. nov. 28.)
- [18] Django Documentation: *Migrations*, <https://docs.djangoproject.com/en/stable/topics/migrations/> (2024. nov. 28.)
- [19] Django Documentation: *Built-in template tags and filters*, <https://docs.djangoproject.com/en/stable/ref/templates/builtins/> (2024. dec. 5.)
- [20] Django Channels: *Channels 4.2.0 Documentation*, <https://channels.readthedocs.io/en/stable/> (2024. nov. 28.)
- [21] Django Channels: *Consumers*, <https://channels.readthedocs.io/en/stable/topics/consumers.html> (2024. nov. 28.)
- [22] Django Channels: *Channel Layers*, https://channels.readthedocs.io/en/stable/topics/channel_layers.html (2024. nov. 28.)
- [23] Django Channels: *Deploying*, <https://channels.readthedocs.io/en/stable/deploying.html#daphne> (2024. nov. 28.)
- [24] Django Community: *Django Közösség*, <https://www.djangoproject.com/community/> (2024. nov. 28.)
- [25] SQLite: *SQLite Documentation*, <https://www.sqlite.org/docs.html> (2024. nov. 28.)
- [26] Django Documentation: *Databases*, <https://docs.djangoproject.com/en/stable/topics/db/> (2024. nov. 28.)
- [27] MDN: *HTML: HyperText Markup Language*, <https://developer.mozilla.org/en-US/docs/Web/HTML> (2024. nov. 28.)
- [28] MDN: *CSS: Cascading Style Sheets*, <https://developer.mozilla.org/en-US/docs/Web/CSS> (2024. nov. 28.)
- [29] Wikipedia: *Cascading Style Sheets*, https://en.wikipedia.org/wiki/Cascading_Style_Sheets (2024. nov. 28.)

- [30] Bootstrap: *Get started with Bootstrap*, <https://getbootstrap.com/docs/5.3/getting-started/introduction/> (2024. nov. 28.)
- [31] MDN: *Working with JSON - Learn web development*, <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON> (2024. nov. 28.)
- [32] MDN: *JavaScript*, <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (2024. nov. 28.)
- [33] MDN: *Fetching data from the server - Learn web development*, https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Fetching_data (2024. nov. 28.)
- [34] MDN: *XMLHttpRequest - Web APIs*, <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest> (2024. nov. 28.)
- [35] Start Bootstrap: *Shop Homepage Template*, <https://startbootstrap.com/template/shop-homepage> (2024. dec. 2.)
- [36] Start Bootstrap: *Shop Item Template*, <https://startbootstrap.com/template/shop-item> (2024. dec. 2.)
- [37] Pytest Documentation, <https://docs.pytest.org> (2024. nov. 28.)
- [38] Coverage.py – Coverage.py 7.6.8 Documentation, <https://coverage.readthedocs.io> (2024. nov. 28.)