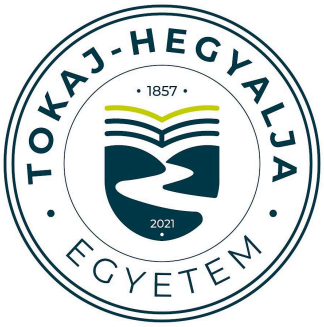

OMT

-

Ingatlankezelő Szoftver

Zöld Pont Hálózat KFT.

Gyakorlat

Készítette:

Aros Damján BSc - EFEW32

Tán Gergő BSc -BLCL2O

Ródé Martin BSc - DRPPXL

Sárospatak, 2025

Tartalomjegyzék

5. Prototípus implementáció

5.1 Főbb komponensek kódstruktúrája

5.2 Skeleton implementáció részletei

5.3 Kritikus útvonalak működése

6. Tesztelési terv

6.1 Tesztstratégia és keretrendszerek

6.2 Tesztesetek és forgatókönyvek

6.3 Tesztkörnyezet és automatizálás

7. Dokumentáció és forráskód

7.1 Leadandó anyagok listája

7.2 Verziókövetés és repository struktúra

7.3 API dokumentáció és adatbázis séma

1. Bevezetés

1.1 Dokumentum célja

Ez a dokumentum az **Ingatlankezelő Szoftver** objektumorientált tervezési fázisát rögzíti, az OMT (Object Modeling Technique) módszertan alapján. A dokumentum célja, hogy részletesen leírja:

- A rendszer **vezérlési modelljét**,
- Az **osztályok közötti kapcsolatok** implementációs módját,
- A **modulok szervezését**,
- A **prototípus implementációját**,
- A **tesztelési stratégiát**.

1.2 Projekt áttekintése

Az Ingatlankezelő Szoftver egy webes platform, amely lehetővé teszi:

- Ingatlanhirdetések **feladását és moderálását**,
- **Keresést** szöveges és térképes szűrőkkel,
- **Üzenetküldést** felhasználók között,
- **Prémium előfizetéseket** Stripe integrációval.

1.3 Felhasználói szerepkörök

Szerepkör	Leírás
Vendég	Csak kereshet, nem adhat fel hirdetést.
Regisztrált	Hirdetést adhat fel, üzenetet küldhet.
Prémium	Korlátlan hirdetés, statisztikák.

Admin	Moderálás, felhasználókezelés.

---

2. Rendszervezérlés tervezése

2.1 Vezérlési modell

A rendszer **eseményvezérelt architektúrát (EDA)** használ, a következő komponensekkel:

- **Aktív objektumok** (folyamatokat kezelnek):
  - `UserSessionController` (bejelentkezés/kilépés),
  - `ListingModerator` (automatikus és kézi moderálás),
  - `PaymentProcessor` (Stripe tranzakciók).
- **Passzív objektumok** (adattárolás):
  - `Listing`, `User`, `Message`.
- **Vezérlő objektumok** (koordináció):

```
public class SystemOrchestrator {
    private KafkaProducer kafkaProducer;

    public void handleEvent(Event event) {
        kafkaProducer.send("system-events", event);
    }

      }
```

2.2 Üzenetváltási mechanizmusok

Példa egy hirdetésfeladási folyamatra:

1. **Felhasználó** → `ListingController.submitListing()`
2. `ListingController` → `ModerationService.validateListing()`
3. `ModerationService` → `NotificationService.sendModerationResult()`

---

3. Osztálykapcsolatok implementációja

3.1 Főbb kapcsolatok

Kapcsolat	Típus	Implementáció
User-Listing	1:N	`User` tartalmaz `List<Listing>`
Listing-Image	1:N	JPA `@OneToMany` annotáció
User-Message	M:N	Külön `Message` tábla `sender_id` és `receiver_id` mezőkkel

3.2 Implementációs osztályok

- **Adatelérési réteg**:

```
@Repository
public class ListingRepository {
    public List<Listing> findByPriceRange(double min, double max) { ... }
```

Külső API integrációk:

```
class GeocodingProxy {
    async getCoordinates(address: string): Promise<LatLng> {
        return fetchGoogleMapsAPI(address);
    }

}
```

---

4. Modulszervezés

4.1 Főbb csomagok

```
src/
├── core/            # Üzleti logika
│   ├── model/       # Entitások
│   └── service/     # Szolgáltatások
├── api/             # REST végpontok
├── persistence/     # Adatbázis kapcsolat
└── integration/     # Külső API-k (Google Maps, Stripe)
```

4.2 Függőségi gráf

```
api → core → persistence
core → integration
```

---

5. Prototípus implementáció

5.1 Főbb komponensek

Flask alapú backend prototípus

```
from flask import Flask, jsonify, request

app = Flask(__name__)

# Dummy adatbazis
felhasznalok = [
    {"id": 1, "nev": "Kiss Bela", "email": "kiss@example.com"}
]

hirdetesek = [
    {
        "id": 1,
        "felhasznalo_id": 1,
        "leiras": "Budapesti 3 szobás lakás",
        "ar": 55000000,
        "ingatlan": {
```

```
        "cim": "Budapest, Petőfi u. 12",
        "alapterulet": 65,
        "szobaszam": 3
    }
  }
]


@app.route("/hirdetesek", methods=["GET"])
def list_hirdetesek():
  return jsonify(hirdetesek)
@app.route("/hirdetesek", methods=["POST"])
def uj_hirdetes():
  data = request.json
  data["id"] = len(hirdetesek) + 1
  hirdetesek.append(data)
  return jsonify({"status": "ok", "id": data["id"]}), 201
if __name__ == "__main__":
  app.run(debug=True)
```

---

6. Tesztelési terv

6.1 Tesztstratégia

Teszt Típus	Keretrendszer	Lefedettség
Egységteszt	JUnit	80%+
Integrációs	Postman	Fő API végpontok
E2E	Cypress	Regisztráció → Hirdetésfeladás

6.2 Tesztesetek

1. **Hirdetés létrehozás**
    - **Input**: Érvényes cím, ár, leírás
    - **Elvárt eredmény**: `HTTP 201 Created`
2. **Fizetési folyamat**
    - **Teszteset**: Sikertelen kártyatranzakció
    - **Elvárt viselkedés**: Rollback az adatbázisban

---

7. Dokumentáció és forráskód

7.1 Leadandó anyagok

1. **Technikai dokumentáció**:
    - Osztálydiagramok (PlantUML)
    - API specifikáció (Swagger/OpenAPI)
2. **Forráskód**:
    - Java (GitHub repository)
    - React frontend
3. **Tesztelési anyagok**:
    - Tesztesetek (Gherkin)

7.2 Verziókövetés

```
git/
├── main      # Stabil verzió
├── develop   # Fejlesztői ág
└── feature/* # Jellemzők külön ágon
```

OMT Objektummodell (elemzés alapján)

Az OMT objektum tervezés első lépése az **osztályok és azok kapcsolatai**. Ezek alapján az alábbi főbb osztályok azonosíthatók az ingatlanos weboldalból:

Fő osztályok:

Felhasználó
Ingatlan
Hirdetés
Kép
Kategória (pl. lakás, ház, iroda)
Helyszín (pl. város, irányítószám)
Admin
Üzenet (kapcsolatfelvételhez)

Kapcsolatok:

Egy **Felhasználó** több **Hirdetést** adhat fel.
Egy **Hirdetés** egy adott **Ingatlanhoz** kapcsolódik.
Egy **Ingatlannak** több **Képe** lehet.
Egy **Hirdetés** egy **Kategóriához** és egy **Helyszínhez** tartozik.
Egy **Felhasználó** írhat **Üzenetet** más hirdetőnek.