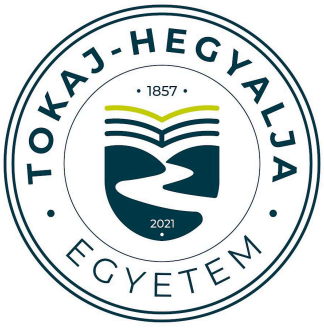

OMT

-

Ingatlankezelő Szoftver

Zöld Pont Hálózat KFT.

Gyakorlat

Készítette:

Aros Damján BSc - EFEW32

Tán Gergő BSc -BLCL2O

Ródé Martin BSc - DRPPXL

Sárospatak, 2025

Tartalomjegyzék

1. Bevezetés

1.1 Dokumentum célja

1.2 Projekt áttekintése

1.3 Felhasználói szerepkörök

2. Rendszervezérlés tervezése

2.1 Vezérlési modell

2.2 Aktív és passzív objektumok

2.3 Üzenetváltási mechanizmusok

2.4 Vezérlő objektumok specifikációja

3. Osztálykapcsolatok implementációja

3.1 Főbb kapcsolatok és implementációs stratégiák

3.2 Asszociációk, aggregációk, kompozíciók

3.3 Implementációs osztályok és interfészek

4. Modulszervezés és csomagstruktúra

4.1 Főbb modulok és felelősségeik

4.2 Csomagdiagram és függőségek

4.3 Kommunikációs protokollok

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

A jelen dokumentum célja, hogy részletesen bemutassa az Ingatlankezelő Szoftver objektumorientált tervezési folyamatát az OMT (Object Modeling Technique) módszertan keretein belül. A leírás átfogó képet ad a rendszer strukturális és vezérlési felépítéséről, valamint az egyes komponensek közötti kapcsolatok működéséről. A dokumentum célja továbbá az, hogy elősegítse a fejlesztés egységes szemléletét, és alapot biztosítson a későbbi implementációs és tesztelési fázisokhoz. Az olvasó betekintést nyerhet a rendszer moduláris felépítésébe, a komponensek együttműködésébe, valamint az alkalmazott technológiai megoldásokba. Emellett a dokumentum hozzájárul a fejlesztői csapat és az érdekeltek közötti hatékony kommunikációhoz, biztosítva, hogy minden érintett fél ugyanazt az architektúrát értelmezze és alkalmazza. A dokumentáció végső célja tehát egy olyan strukturált és konzisztens tervezési alap megteremtése, amely támogatja a projekt sikeres megvalósítását.

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

1.2 Projekt áttekintése

A projekt célja egy korszerű, felhasználóbarát ingatlankezelő rendszer megtervezése és kidolgozása, amely hatékonyan támogatja a különböző ingatlankezeléssel kapcsolatos adminisztratív és nyilvántartási feladatokat. A rendszer lehetővé teszi a felhasználók számára az ingatlanok adatainak naprakész kezelését, a bérlői információk nyilvántartását, a szerződések és pénzügyi tranzakciók követését, valamint egyéb releváns műveletek elvégzését. A projekt során kiemelt figyelmet fordítunk arra, hogy a rendszer megfeleljen a mai kor elvárásainak mind funkcionalitás, mind skálázhatóság, mind pedig biztonság tekintetében.

Az ingatlankezelő alkalmazás fejlesztése során törekszünk arra, hogy a különféle felhasználói szerepkörök (pl. adminisztrátor, ügyintéző, könyvelő) számára jól elkülönített, mégis integrált funkcionalitást biztosítsunk. A rendszer tervezésénél fontos szempont az átlátható felhasználói felület, a hatékony adatkezelés, valamint az egyszerűen bővíthető moduláris architektúra. Emellett biztosítani kívánjuk az adatok védelmét és a jogosultságkezelés korszerű elveken alapuló megvalósítását.

A projekt során az OMT módszertan alkalmazásával modellezzük az objektumokat, azok attribútumait, viselkedését és egymással való kapcsolatait. A cél egy jól strukturált, hosszú távon is fenntartható rendszerterv kidolgozása, amely megalapozza a szoftver hatékony fejlesztését, bevezetését és karbantartását. A rendszer kiemelten fontos szerepet tölthet be olyan cégek vagy intézmények életében, amelyek több ingatlan kezeléséért felelősek, és szükségük van egy megbízható, automatizált adminisztrációs eszközre.

2. Rendszervezérlés tervezése

A rendszervezérlés célja, hogy biztosítsa az egyes komponensek közötti hatékony együttműködést és az eseményekre való gyors és megbízható reagálást. A vezérlés központi szerepet tölt be az alkalmazás működésében, mivel ez határozza meg az események feldolgozásának menetét, a műveletek sorrendjét, valamint a különböző objektumok és szolgáltatások közötti kommunikációt.

2.1 Vezérlési modell

A rendszer egy eseményvezérelt architektúrára (Event-Driven Architecture – EDA) épül, amely lehetővé teszi, hogy a komponensek egymástól függetlenül reagáljanak a különféle eseményekre, így biztosítva a nagyfokú skálázhatóságot, modularitást és rugalmasságot.

Az eseményvezérelt rendszer három fő komponenscsoportra oszlik:

1. **Aktív objektumok:**
   Ezek az objektumok különböző folyamatokat irányítanak, aktívan reagálnak a rendszerben bekövetkező eseményekre, és gyakran háttérszolgáltatásként működnek.
   - **UserSessionController:** Felelős a felhasználói munkamenetek kezeléséért (bejelentkezés, kilépés, munkamenet-érvényesítés).
   - **ListingModerator:** Kezeli a hirdetések moderálását (AI-alapú előszűrés + admin felülvizsgálat).
   - **PaymentProcessor:** Stripe integráción keresztül kezeli a fizetési tranzakciókat.
2. **Passzív objektumok:**
   Ezek az objektumok elsősorban adatok tárolására szolgálnak, viselkedésük minimális.
   - **Listing**: Ingatlanhirdetések adatait tárolja (cím, leírás, ár, státusz).
   - **User:** Felhasználói fiókadatok (név, e-mail, jogosultságok).
   - **Message:** Üzenetek és értesítések adatmodellje.
3. **Vezérlő objektumok:**
   Ezek koordinálják az alrendszerek közötti kommunikációt.
   - **SystemOrchestrator:** Központi Kafka-alapú eseménykezelő.

```
public class SystemOrchestrator {
private KafkaProducer kafkaProducer;
public void handleEvent(Event event) {
kafkaProducer.send("system-events", event);
}
      }
```

2.2 Üzenetváltási mechanizmusok

A kommunikáció szinkron vagy aszinkron üzenetekkel történik, az alábbi példa egy hirdetésfeladási folyamatot mutat be:

1. **Felhasználó** → ListingController.submitListing()
    - A felhasználó elküldi a hirdetést.
2. **ListingController** → ModerationService.validateListing()
    - A moderálási szolgáltatás ellenőrzi a tartalmat.
3. **ModerationService** → NotificationService.sendModerationResult()
    - Az eredményt értesítésként továbbítja a felhasználónak.

Az EDA előnyei: lazán csatolt komponensek, skálázhatóság, könnyű bővíthetőség.

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

Felhasznalo.java

```
public class Felhasznalo {
    private int id;
    private String nev;
    private String email;

    public Felhasznalo(int id, String nev, String email) {
        this.id = id;
        this.nev = nev;
        this.email = email;
    }

    // Getterek és setterek
    public int getId() { return id; }
    public String getNev() { return nev; }
    public String getEmail() { return email; }

    public void setId(int id) { this.id = id; }
    public void setNev(String nev) { this.nev = nev; }
    public void setEmail(String email) { this.email = email; }
}
```

Ingatlan.java

```
public class Ingatlan {
    private String cim;
    private int alapterulet;
    private int szobaszam;

    public Ingatlan(String cim, int alapterulet, int szobaszam) {
        this.cim = cim;
        this.alapterulet = alapterulet;
        this.szobaszam = szobaszam;
    }

    public String getCim() { return cim; }
    public int getAlapterulet() { return alapterulet; }
    public int getSzobaszam() { return szobaszam; }

    public void setCim(String cim) { this.cim = cim; }
        public void setAlapterulet(int alapterulet) { this.alapterulet =
alapterulet; }
    public void setSzobaszam(int szobaszam) { this.szobaszam = szobaszam; }
}
```

Feladatok.java

```
public class Feladatok {
   private int task_id;
   private int user_id;
   private String nev;
   private String leiras;
   private Date hatarido;

    public Feladatok(int task_id, int user_id, String nev, String leiras, Date
hatarido) {
       this.task_id = task_id;
       this.user_id = user_id;
       this.nev = nev;
       this.leiras = leiras;
       this.hatarido = hatarido;}

   public int getTask_id() { return task_id; }
   public void setTask_id(int task_id) { this.task_id = task_id; }
   public int getUser_id() { return user_id; }
   public void setUser_id(int user_id) {this.user_id = user_id; }
```

```
   public String getNev() { return nev; }
   public void setNev(String nev) { this.nev = nev; }
   public String getLeiras() { return leiras; }
   public void setLeiras(String leiras) { this.leiras = leiras; }
   public Date getHatarido() { return hatarido; }
   public void setHatarido(Date hatarido) { this.hatarido = hatarido; }
}
```

Ugyfelek.java

```
public class Ugyfelek {

   private int ugyfel_id;
   private int user_id;
   private String nev;
   private String email;
   private String telszam;
   private String lakcim;
   private String varos;

     public Ugyfelek(int ugyfel_id, int user_id, String nev, String email,
String telszam, String lakcim, String varos) {
       this.ugyfel_id = ugyfel_id;
       this.user_id = user_id;
       this.nev = nev;
       this.email = email;
       this.telszam = telszam;
       this.lakcim = lakcim;
       this.varos = varos;
   }

   public int getUgyfel_id() { return ugyfel_id; }
   public void setUgyfel_id(int ugyfel_id) { this.ugyfel_id = ugyfel_id; }
   public int getUser_id() { return user_id; }
   public void setUser_id(int user_id) { this.user_id = user_id; }
   public String getNev() { return nev; }
   public void setNev(String nev) { this.nev = nev; }
   public String getEmail() { return email; }
   public void setEmail(String email) { this.email = email; }
   public String getTelszam() { return telszam; }
   public void setTelszam(String telszam) { this.telszam = telszam; }
   public String getLakcim() { return lakcim; }
   public void setLakcim(String lakcim) { this.lakcim = lakcim; }
   public String getVaros() { return varos; }
   public void setVaros(String varos) { this.varos = varos; }
}
```

Hirdetes.java

```
public class Hirdetes {
    private int id;
    private int felhasznaloId;
    private String leiras;
    private int ar;
    private Ingatlan ingatlan;

    public Hirdetes(int id, int felhasznaloId, String leiras, int ar, Ingatlan
ingatlan) {
        this.id = id;
        this.felhasznaloId = felhasznaloId;
        this.leiras = leiras;
        this.ar = ar;
        this.ingatlan = ingatlan;
    }

    public int getId() { return id; }
    public int getFelhasznaloId() { return felhasznaloId; }
    public String getLeiras() { return leiras; }
    public int getAr() { return ar; }
    public Ingatlan getIngatlan() { return ingatlan; }

    public void setId(int id) { this.id = id; }
      public void setFelhasznaloId(int felhasznaloId) { this.felhasznaloId =
felhasznaloId; }
    public void setLeiras(String leiras) { this.leiras = leiras; }
    public void setAr(int ar) { this.ar = ar; }
    public void setIngatlan(Ingatlan ingatlan) { this.ingatlan = ingatlan; }
}
```

`Adatbazis.java` (Dummy adatbázis)

```
import java.util.ArrayList;
import java.util.List;

public class Adatbazis {
    public static List<Felhasznalo> felhasznalok = new ArrayList<>();
    public static List<Hirdetes> hirdetesek = new ArrayList<>();

    static {
        // Dummy adat inicializálása
        felhasznalok.add(new Felhasznalo(1, "Kiss Béla", "kiss@example.com"));

        Ingatlan ingatlan = new Ingatlan("Budapest, Petőfi u. 12", 65, 3);
             hirdetesek.add(new Hirdetes(1, 1, "Budapesti 3 szobás lakás",
55000000, ingatlan));
    }

    public static void ujHirdetes(Hirdetes hirdetes) {
        hirdetes.setId(hirdetesek.size() + 1);
        hirdetesek.add(hirdetes);
    }

    public static List<Hirdetes> getHirdetesek() {
        return hirdetesek;
    }

    public static boolean torolHirdetes(int id) {
        return hirdetesek.removeIf(h -> h.getId() == id);
    }
}
```

`Session.java`

```
public class Session {

   private String session_id;
   private int user_id;
   private Date login_time;
   private Date last_activity;
   private String ip;

     public Session(String session_id, int user_id, Date login_time, Date
last_activity, String ip) {
       this.session_id = session_id;
```

```
        this.user_id = user_id;
        this.login_time = login_time;
        this.last_activity = last_activity;
        this.ip = ip;
    }

    public String getSession_id() { return session_id; }
        public void setSession_id(String session_id) { this.session_id =
session_id; }
    public int getUser_id() { return user_id; }
    public void setUser_id(int user_id) { this.user_id = user_id; }
    public Date getLogin_time() { return login_time; }
     public void setLogin_time(Date login_time) { this.login_time = login_time;
}
    public Date getLast_activity() { return last_activity; }
      public void setLast_activity(Date last_activity) { this.last_activity =
last_activity; }
    public String getIp() { return ip; }
    public void setIp(String ip) { this.ip = ip; }
}
```

`Main.java` (Tesztelés parancssorban – nem REST)

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Aktuális hirdetések:");
        for (Hirdetes h : Adatbazis.getHirdetesek()) {
             System.out.println(h.getLeiras() + " - " + h.getAr() + " Ft - " +
h.getIngatlan().getCim());
        }

        System.out.println("\nÚj hirdetés hozzáadása...");
        Ingatlan ujIngatlan = new Ingatlan("Debrecen, Kossuth utca 5", 75, 4);
           Hirdetes ujHirdetes = new Hirdetes(0, 1, "Debreceni tágas lakás",
48000000, ujIngatlan);
        Adatbazis.ujHirdetes(ujHirdetes);

        System.out.println("\nFrissített hirdetéslista:");
        for (Hirdetes h : Adatbazis.getHirdetesek()) {
              System.out.println(h.getId() + ": " + h.getLeiras() + " - " +
h.getIngatlan().getCim());
        }
    }
}
```

---

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

3. **Tesztelési terv- Frontend:**

Teszteset azonosító	Funkció	Bemenet	Várt kimenet
TC-F01	Hirdetések megjelenítése	Oldal betöltése után automatikusan	Meglévő hirdetések kártyákban jelennek meg
TC-F02	Hirdetés beküldése	Form kitöltése és `Hirdetés létrehozása` gomb megnyomása	Hirdetés megjelenik a listában frissítés nélkül
TC-F03	Hirdetés törlése	`Törlés` gomb megnyomása egy kártyán	Hirdetés eltűnik a listából
TC-F04	Form validáció	Üres mezők beküldése	Böngésző blokkolja a beküldést, nem történik POST
TC-F05	Backend nem elérhető	API szerver le van állítva	Nem jelennek meg hirdetések, hibát jelezhet a konzol
TC-F06	Ár vagy szobaszám negatív	Ár = -100, Szobaszám = -1	Hirdetés nem jön létre (backend oldalon kell validálni)
TC-F07	Mobilnézet kompatibilitás	Megnyitás mobil eszközön vagy kis ablakban	Oldal reszponzív, nem csúszik szét

Teszt-eset azonosító	Funkció	Bemenet	Várt kimenet
TC01	Hirdetések lekérése	GET `/hirdetesek`	JSON list of ads
TC02	Új hirdetés beküldés	POST `/hirdetesek` + JSON body	`201 Created` + ID
TC03	Üres hirdetés	POST `/hirdetesek` üres body	`400 Bad Request` vagy hibaüzenet

---

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.