

**VÁCI SZAKKÉPZÉSI CENTRUM BORONKAY GYÖRGY MŰSZAKI
TECHNIKUM ÉS GIMNÁZIUM**

VIZSGAREMEK

Pék Gergő
Kiss Kolos
Garai Nándor
2025

VÁCI SZAKKÉPZÉSI CENTRUM BORONKAY GYÖRGY MŰSZAKI
TECHNIKUM ÉS GIMNÁZIUM



VIZSGAREMEK

EasyInventory

Konzulens: X Készítette: Pék Gergő
Kiss Kolos
Garai Nándor

Hallgatói nyilatkozat

Alulírottak, ezúton kijelentjük, hogy a szakdolgozat saját, önálló munkánk, és korábban még sehol nem került publikálásra. Szakdolgozatunk a Váci Szakképzési Centrum Boronkay György Műszaki Technikum és Gimnázium Szoftverfejlesztő és tesztelő technikus képzésén készítettük. Tudomásul vesszük, hogy szakdolgozatunkat a Váci Szakképzési Centrum Boronkay György Műszaki Technikum és Gimnázium tárolja.

Pék Gergő

Kis Kolos

Garai Nándor

Konzultációs lap

Vizsgázók neve: Pék Gergő

Kiss Kolos

Garai Nándor

Szakdolgozat címe: EasyInventory

Program által nyújtott szolgáltatások:

- Raktárak és telephelyek kezelése
- Mértékegységek és terméktípusok kezelése
- Raktárak tartalmának módosítása műveletek segítségével
- Felhasználók kezelése
- Lokális és rendszerszintű jogosultságok
- Raktárak tartalmáról statisztikák kimutatása
- Keresés raktárakban és telephelyeken
- Raktárak kapacitásának beállítása
- Felhasználói felület asztali ,webes, mobilos platformon

Sorszám	A konzultáció időpontja	A konzulens aláírása
1.	2024. október 11.	
2.	2024. november 15.	
3.	2024. december 13.	
4.	2025. január 17.	
5.	2025. február 14.	
6.	2025. március 14.	

A szakdolgozat beadható:
Vác, 2025.

Konzulens

A szakdolgozatot átvettetem:
Vác, 2025.

A szakképzést folytató intézmény felelőse

Tartalom

1 Konzultációs lap	5
2 Munka	9
2.1 Munkamegosztás	10
2.1.1 Pék Gergő	10
2.1.2 Kiss Kolos	10
2.1.3 Garai Nándor	10
2.2 Munka folyamata	10
2.3 Használt fejlesztői eszközök	11
3 API Könyvtár Dokumentáció	13
3.1 Kapcsolat kezelés	14
3.1.1 Kapcsolódás a szerverhez	14
3.1.2 Kijelentkezés	14
3.2 Egységek kezelése	15
3.2.1 Egység osztály	15
3.2.2 Egységek kezelése	15
3.3 Felhasználók kezelése	16
3.3.1 Felhasználó osztály	16
3.3.2 Felhasználók kezelése	16
3.3.3 Felhasználók engedélyeinek kezelése	17
3.4 Telephelyek kezelése	17
3.4.1 Telephely osztály	17
3.4.2 Telephelyek kezelése	18
3.5 Raktárak kezelése	18
3.5.1 Raktár osztály	18
3.5.2 Raktárak kezelése	19
3.6 Cikkek kezelése	20
3.6.1 Cikk osztály	20
3.6.2 Cikkek kezelése	20
3.7 Műveletek kezelése	21
3.7.1 Művelet osztály	21
3.7.2 Műveletek kezelése	21
3.8 Keresés	22
3.8.1 ItemStack osztály	22
3.8.2 Keresés	22
4 Kód Dokumentáció	23
4.1 Felépítés	24
4.2 Adatbázis	25
4.2.1 Áttekintés	25
4.2.2 untis	27

4.2.3	item_types	27
4.2.4	warehouses	27
4.2.5	storages	28
4.2.6	item_stacks	28
4.2.7	storage_limits	29
4.2.8	operations	29
4.2.9	operation_items	30
4.2.10	users	30
4.2.11	authorization	31
4.2.12	global_authorization	31
4.2.13	authentication_token	32
4.2.14	users_view	32
4.2.15	authorization_view	32
4.2.16	authentication_token_view	33
4.2.17	warehouses_view	33
4.2.18	storages_view	33
4.2.19	item_types_view	33
4.2.20	units_view	34
4.2.21	operations_view	34
4.2.22	operation_items_view	35
4.2.23	item_stacks_view	36
4.3	Api szerver	37
4.3.1	Felépítés	37
4.3.2	ApiRouter.php	38
4.3.3	APIUtils.php	39
4.3.4	DB.php	42
4.3.5	BaseTrait.php	42
4.3.6	Logger.php	42
4.3.7	Settings.php	43
4.3.8	Authentication.php	43
4.4	Web frontend	44
4.4.1	Áttekintés	44
4.4.2	Építés	44
4.4.3	Template osztály	44
4.4.4	View osztály	46
4.4.5	ListView osztály	46
4.4.6	ListEditView osztály	47
4.4.7	SelectView osztály	47
4.4.8	DialogView osztály	48
4.4.9	EditDialog osztály	48
4.5	API könyvtár	49

4.5.1	Áttekintés	49
4.5.2	HTTPConnector osztály	49
4.6	Asztali kliens	50
4.6.1	Bevezetés	50
4.6.2	Worker osztály	50
4.6.3	EasyInventoryDesktop osztály	50
4.6.4	BasicView osztály	52
4.6.5	SelectView osztály	52
4.6.6	CardView osztály	52
4.6.7	EditableView osztály	53
4.6.8	LocalEditableView osztály	53
4.6.9	FormDialog osztály	55
4.6.10	EditDialog osztály	55
4.7	Mobil kliens	56
4.7.1	Áttekintés	56
4.7.2	MainActivity osztály	56
4.7.3	LoggedInActivity osztály	56
4.7.4	Worker osztály	57
4.7.5	BaseEditActivity osztály	59
4.7.6	LocalEditActivity osztály	59
4.7.7	PaginatedListActivity osztály	61
4.7.8	BaseListActivity osztály	61
4.7.9	BaseListFragment osztály	63
4.7.10	LocalListFragment osztály	64
4.7.11	SelectableArrayAdapter osztály	66
4.8	Dokumentáció	67
4.8.1	Áttekintés	67
4.8.2	Felhasználói dokumentáció	67
4.8.3	Kód dokumentáció	67
4.8.4	Teszt dokumentáció	67
4.8.5	Kód beillesztő	68
4.8.6	Építés	68
5	Teszt Dokumentáció	69
5.1	API szerver	70
5.1.1	Áttekintés	70
5.1.2	E2E tesztek	70
5.2	Web frontend	71
5.2.1	Áttekintés	71
5.2.2	E2E tesztek	71
5.3	Asztali alkalmazás	72
5.3.1	Áttekintés	72

5.3.2	E2E tesztek	72
5.3.3	NodeMatcher osztály	73
5.4	Mobil alkalmazás	76
5.4.1	Áttekintés	76
5.4.2	E2E tesztek	76
6	Felhasználói Dokumentáció	77
6.1	Bevezetés	78
6.1.1	Fogalmak	78
6.2	Telepítés	81
6.2.1	Server telepítés	81
6.2.2	Asztali kliens telepítés	82
6.2.3	Mobil kliens telepítés	82
6.3	Adminisztráció	83
6.3.1	Arhivált adatok törlése	83
6.3.2	Eseménynapló kezelése	83
6.3.3	Felhasználó kezelés	85
6.3.4	Engedélyek	86
6.3.5	Beállítások	87
6.4	Használat	90
6.4.1	Bejelentkezés	90
6.4.2	Felhasználó kezelés	91
6.4.3	Egység kezelés	93
6.4.4	Árucikkek kezelése	95
6.4.5	Telephely kezelés	97
6.4.6	Raktár kezelés	99
6.4.7	Műveletek	101
6.4.8	Keresés	102
6.5	Használati példák	103
6.5.1	Fatelep	103
6.5.2	Bevásárló központ	104
7	Továbbfejlesztési lehetőségek	105
7.1	Adminisztráció	106
7.1.1	Integráció monitorozó rendszerekkel	106
7.1.2	Központi beléptető rendszer	106
7.1.3	Webhook-ok	106
7.2	Statisztika	106
7.2.1	Kördiagram telephely/raktár tartalmáról	106
7.2.2	Különböző raktárak/telephelyek összehasonlítása	107
7.3	Használat	107
7.3.1	Térkép a raktárról és benne a tárgyakról	107

7.3.2	Rendszeres művelet order	107
7.3.3	Mértékegység átváltás	107
7.3.4	Raktártípusok	108
7.3.5	Riasztás beállítás	108
7.3.6	QR kód olvasó	108
8	Felhasznált irodalom	109
9	Mellékletek	111
9.1	Felhasznált könyvtárak	112
9.2	Online elérhetőség	112

Munka

2.1 Munkamegosztás

2.1.1 Pék Gergő

Csoportvezetés, API megtervezése és elkészítése, mobil és asztali alkalmazások létrehozása, webes frontend és API szerver architektúrájának tervezése, adatbázis megtervezése és elkészítése, dokumentáció megírása, stílusának tervezése és kód minőségének ellenőrzése.

2.1.2 Kiss Kolos

API tesztek írása, webes frontend elkészítése, felhasználói dokumentáció megírása és nyelvi helyességének ellenőrzése, design elemek (ikonok, logo) megrajzolása, nyelvi fájlok (angol, magyar) elkészítése.

2.1.3 Garai Nándor

Webes és API tesztek írása, API szerver adatbázis kezelő részének implementálása, web frontend design megtervezése, tesztadatok generálása és bemutató készítése.

2.2 Munka folyamata

A fejlesztés teszt alapú megközelítéssel zajlott. Először az API készült el, majd párhuzamosan dolgoztunk a három felületen (mobil, asztali, web) és a hozzájuk tartozó teszteken. Végül a dokumentáció elkészítése és a végső simítások következtek.

2.3 Használt fejlesztői eszközök

- Netbeans
- VSCode
- Android Studio
- PHPMyAdmin
- Inkscape
- XeLatex

API Könyvtár

Dokumentáció

3.1 Kapcsolat kezelés

3.1.1 Kapcsolódás a szerverhez

Az API szerverhez való kapcsolódást a `connect` funkció hajtja végre. A funkció egy `API` objektumot ad vissza, aminek a segítségével kommunikálhatunk a szerverrel.

```
public class API {  
    public static API connect(HTTPConnector connector, String url,  
        String username, String password) throws IOException,  
        APIException, JSONException { ... }  
    ...  
}
```

3.1.2 Kijelentkezés

A kijelentkezéshez a `logout` metódust kell meghívni. Ezt követően nem lehetséges az `API` objektum további használata.

```
public class API {  
    public void logout() throws IOException, APIException,  
        JSONException { ... }  
    ...  
}
```

3.2 Egységek kezelése

3.2.1 Egység osztály

Az egység adatainak tárolásához a `Unit` osztály használható.

```
public class Unit implements ToJSON {
    public final String id;
    public final String name;
    public final long deleted;
    public Unit(String id, String name, long deleted) { ... }
    public Unit(String id, String name) { ... }
    public Unit(JSONObject o) throws JSONException { ... }
    @Override
    public JSONObject toJSON() { ... }
    @Override
    public int hashCode() { ... }
    @Override
    public boolean equals(Object obj) { ... }
}
```

3.2.2 Egységek kezelése

A cikkek kezeléséhez az `API` osztályban található funkciókat használhatjuk.

```
public class API {
    public Unit[] getUnits(String query, int offset, int len, boolean
        archived) throws IOException, APIException, JSONException {
        ...
    }
    public Unit getUnit(String id) throws IOException, APIException,
        JSONException { ... }
    public boolean putUnit(String id, String name, boolean create,
        boolean update) throws IOException, APIException,
        JSONException { ... }
    public boolean deleteUnit(String id) throws IOException,
        APIException { ... }
    public boolean moveUnit(String id, String new_id) throws
        IOException, APIException, JSONException { ... }
    ...
}
```

3.3 Felhasználók kezelése

3.3.1 Felhasználó osztály

Az felhasználók adatainak tárolásához a `User` osztály használható.

```
public class User implements ToJSON {
    public final String id;
    public final String name;
    public final String password;
    public final User manager;
    public User(String id, String name, String password, User manager)
        { ... }
    public User(JSONObject o) throws JSONException { ... }
    @Override
    public JSONObject toJSON() { ... }
    @Override
    public int hashCode() { ... }
    @Override
    public boolean equals(Object obj) { ... }
}
```

3.3.2 Felhasználók kezelése

A felhasználók kezeléséhez az `API` osztályban található funkciókat használhatjuk.

```
public class API {
    public UserInfo getUserinfo() throws IOException, APIException,
        JSONException { ... }
    public User[] getUsers(String query, int offset, int len) throws
        IOException, APIException, JSONException { ... }
    public User getUser(String id) throws IOException, APIException,
        JSONException { ... }
    public boolean putUser(String id, String name, String password,
        String manager_id, boolean create, boolean update) throws
        IOException, APIException, JSONException { ... }
    public boolean deleteUser(String id) throws IOException,
        APIException { ... }
    public boolean moveUser(String id, String new_id) throws
        IOException, APIException, JSONException { ... }
    public void changePassword(String old_password, String
        new_password) throws IOException, APIException, JSONException
        { ... }
    ...
}
```

3.3.3 Felhasználók engedélyeinek kezelése

A felhasználók engedélyeinek kezeléséhez az API osztályban található funkciókat használhatjuk.

```
public class API {
    public SystemAuthorization getSystemAuthorization(String user)
        throws IOException, APIException, JSONException { ... }
    public LocalAuthorization getLocalAuthorization(String user,
        String warehouse) throws IOException, APIException,
        JSONException { ... }
    public boolean grantSystemAuthorization(String user, String
        authorization) throws IOException, APIException { ... }
    public boolean revokeSystemAuthorization(String user, String
        authorization) throws IOException, APIException { ... }
    public boolean grantLocalAuthorization(String user, String
        warehouse, String authorization) throws IOException,
        APIException { ... }
    public boolean revokeLocalAuthorization(String user, String
        warehouse, String authorization) throws IOException,
        APIException { ... }
    ...
}
```

3.4 Telephelyek kezelése

3.4.1 Telephely osztály

Az telephelyek adatainak tárolásához a `Warehouse` osztály használható.

```
public class Warehouse implements ToJSON {
    public final String id;
    public final String name;
    public final String address;
    public final long deleted;
    public Warehouse(String id, String name, String address, long
        deleted) { ... }
    public Warehouse(String id, String name, String address) { ... }
    public Warehouse(JSONObject o) throws JSONException { ... }
    @Override
    public JSONObject toJSON() { ... }
    @Override
    public boolean equals(Object obj) { ... }
    @Override
    public int hashCode() { ... }
}
```

3.4.2 Telephelyek kezelése

A telephelyek kezeléséhez az API osztályban található funkciókat használhatjuk.

```
public class API {
    public Warehouse[] getWarehouses(String query, int offset, int
        len, boolean archived) throws IOException, APIException,
        JSONException { ... }
    public Warehouse getWarehouse(String id) throws IOException,
        APIException, JSONException { ... }
    public boolean putWarehouse(String id, String name, String
        address, boolean create, boolean update)
    public boolean deleteWarehouse(String id) throws IOException,
        APIException { ... }
    public boolean moveWarehouse(String id, String new_id) throws
        IOException, APIException, JSONException { ... }
    ...
}
```

3.5 Raktárak kezelése

3.5.1 Raktár osztály

Az raktárak adatainak tárolásához a `Storage` osztály használható.

```
public class Storage implements ToJSON {
    public final String id;
    public final String name;
    public final Warehouse warehouse;
    public final long deleted;
    public Storage(Warehouse warehouse, String id, String name, long
        deleted) { ... }
    public Storage(Warehouse warehouse, String id, String name) { ... }
    public Storage(JSONObject o) throws JSONException { ... }
    @Override
    public JSONObject toJSON() { ... }
    @Override
    public int hashCode() { ... }
    @Override
    public boolean equals(Object obj) { ... }
}
```

3.5.2 Raktárak kezelése

A raktárak kezeléséhez az API osztályban található funkciókat használhatjuk.

```
public class API {  
    public Storage getStorage(String warehouse, String id) throws  
        IOException, APIException, JSONException { ... }  
    public boolean putStorage(String warehouse, String id, String  
        name, boolean create, boolean update) throws IOException,  
        APIException, JSONException { ... }  
    public boolean deleteStorage(String warehouse, String id) throws  
        IOException, APIException { ... }  
    public boolean moveStorage(String warehouse, String id, String  
        new_id) throws IOException, APIException, JSONException { ... }  
    public StorageLimit[] getStorageLimits(String warehouse, String  
        id, String query, int offset, int len) throws IOException,  
        APIException, JSONException { ... }  
    public boolean setStorageLimit(String warehouse, String id,  
        String item, int amount) throws IOException, APIException,  
        JSONException { ... }  
    public StorageCapacity[] getStorageCapacity(String warehouse,  
        String id, String query, int offset, int len) throws  
        IOException, APIException, JSONException { ... }  
    ...  
}
```

3.6 Cikkek kezelése

3.6.1 Cikk osztály

A cikkek adatainak tárolásához a `Item` osztály használható.

```
public class Item implements ToJSON {
    public final String id;
    public final String name;
    public final Unit unit;
    public final long deleted;
    public Item(String id, String name, Unit unit, long deleted) { ... }
    ...
    public Item(String id, String name, Unit unit) { ... }
    public Item(JSONObject o) throws JSONException { ... }
    @Override
    public JSONObject toJSON() { ... }
    @Override
    public int hashCode() { ... }
    @Override
    public boolean equals(Object obj) { ... }
}
```

3.6.2 Cikkek kezelése

A cikkek kezeléséhez az `API` osztályban található funkciókat használhatjuk.

```
public class API {
    public Item[] getItems(String query, int offset, int len, boolean
        archived) throws IOException, APIException, JSONException {
        ...
    }
    public Item getItem(String id) throws IOException, APIException,
        JSONException { ... }
    public boolean putItem(String id, String name, String unit_id,
        boolean create, boolean update) throws IOException,
        APIException, JSONException { ... }
    public boolean deleteItem(String id) throws IOException,
        APIException { ... }
    public boolean moveItem(String id, String new_id) throws
        IOException, APIException, JSONException { ... }
    ...
}
```

3.7 Műveletek kezelése

3.7.1 Művelet osztály

Az műveletek adatainak tárolásához a `Operation` osztály használható.

```
public class Operation implements ToJSON {
    public final String id;
    public final String name;
    public final boolean is_add;
    public final OperationItem[] items;
    public final long deleted;
    public Operation(String id, String name, boolean is_add,
        OperationItem[] items, long deleted) { ... }
    public Operation(String id, String name, boolean is_add,
        OperationItem[] items) { ... }
    public Operation(JSONObject o) throws JSONException { ... }
    @Override
    public JSONObject toJSON() { ... }
    @Override
    public int hashCode() { ... }
    @Override
    public boolean equals(Object obj) { ... }
}
```

3.7.2 Műveletek kezelése

A műveletek kezeléséhez az `API` osztályban található funkciókat használhatjuk.

```
public class API {
    public Operation[] getOperations(String warehouse, String query,
        int offset, int len, boolean archived) throws IOException,
        APIException, JSONException { ... }
    public boolean putOperation(String warehouse, String id, String
        name, boolean is_add, OperationItem[] items) throws
        IOException, APIException, JSONException { ... }
    public boolean cancelOperation(String warehouse, String id)
        throws IOException, APIException { ... }
    public boolean finishOperation(String warehouse, String id)
        throws IOException, APIException { ... }
    ...
}
```

3.8 Keresés

3.8.1 ItemStack osztály

A keresés eredményeinek tárolásához az `ItemStack` osztályt használhatjuk.

```
public class ItemStack implements ToJSON {
    public final Warehouse warehouse;
    public final Storage storage;
    public final Item item;
    public final String lot;
    public final String manufacturer_serial;
    public final int amount;
    public final int available_amount;
    public final int global_serial;
    public ItemStack(Warehouse warehouse, Storage storage, Item item,
                     String lot,
                     String manufacturer_serial, int amount, int
                     available_amount, int global_serial) { ... }
    public ItemStack(JSONObject o) throws JSONException { ... }
    @Override
    public int hashCode() { ... }
    @Override
    public boolean equals(Object obj) { ... }
    @Override
    public JSONObject toJSON() { ... }
}
```

3.8.2 Keresés

A kereséshez az `API` osztályban található funkciókat használhatjuk.

```
public class API {
    public ItemStack[] getCurrentItems(String warehouse, String
                                         storage, String query,
                                         ...
}
```

Kód Dokumentáció

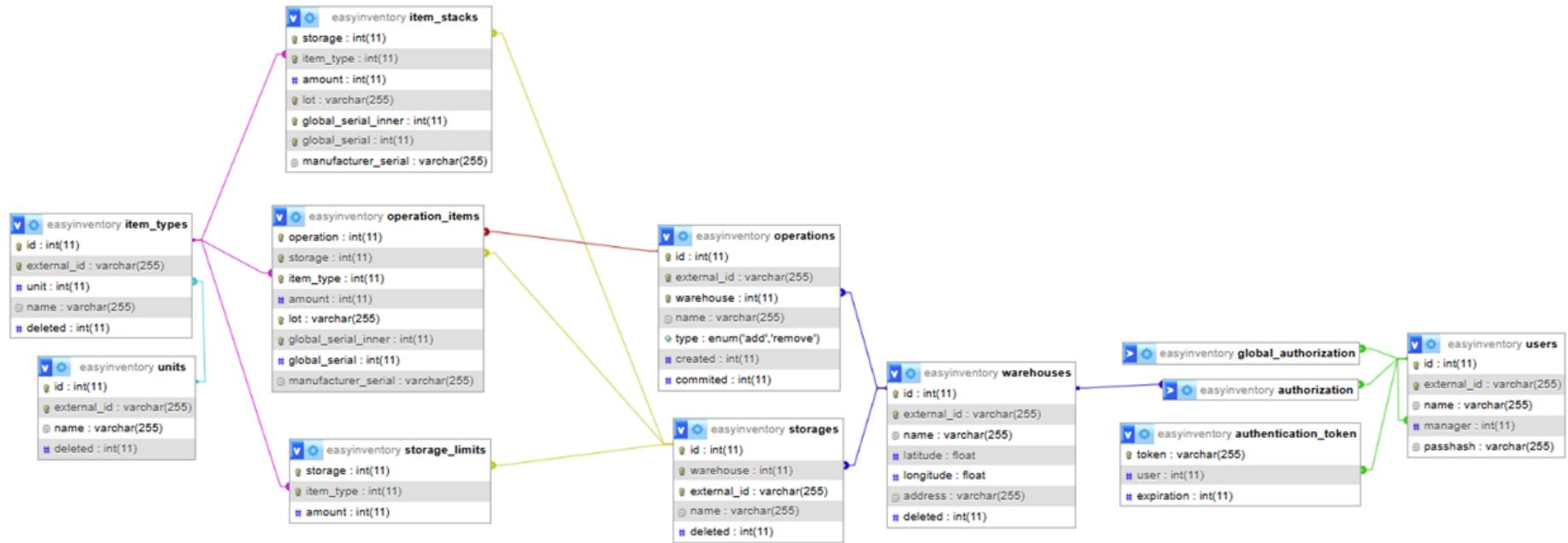
4.1 Felépítés

A gyökérmapában találhatók a `api` , `apitest` , `EasyInventoryAPI` , `EasyInventoryDesktop` , `EasyInventoryMobile` , `doc` , `frontend` és a `lang` mappák. Továbbá ez tartalmazza a `startphp.bat` scriptet, ami elindítja a teszt PHP szervert és a `vscode.bat` fájlt, ami elindítja a VS-Code fejlesztői környezetet. A `lang` mappa tartalmazza a nyelvfájlokat és egy `update.py` scriptet, ami ellenőrzi a nyelvfájlokat és frissíti a programok nyelvfájljait.

4.2 Adatbázis

4.2.1 Áttekintés

A rendszer Mysql relációs adatbázist használ. Az adatbázis létrehozásához szükséges SQL kód a `api/db.sql` fájlban található. A táblákban sokszor megtalálható az `external_id` attributum, amely az API által használt azonosítót tárolja. Erre azért van szükség, mert a valódi azonosító módosításához a táblák nagy részét le kéne zárnai, ami lassítaná a lekéréseket. Az adatbázisban több nézet is található a lekérések egyszerűbbé tételeire. Az alábbi grafikon mutatja a táblák kapcsolatrendszerét:



4.2.2 units

A `units` tábla a mértékegységek tárolásáért felelős.

```
create table if not exists units (
    id int auto_increment not null,
    external_id varchar(255) not null unique,
    name varchar(255) not null,
    deleted int default null,
    primary key(id)
)$
```

4.2.3 item_types

A `item_types` tábla a különböző cikk típusok tárolásáért felelős.

```
create table if not exists item_types (
    id int auto_increment not null,
    external_id varchar(255) not null unique,
    unit int not null,
    name varchar(255) not null,
    deleted int default null,
    primary key(id),
    foreign key(unit) references units(id)
)$
```

4.2.4 warehouses

A `warehouses` tábla a különböző telephelyek tárolásáért felelős.

```
create table if not exists warehouses (
    id int not null auto_increment,
    external_id varchar(255) not null unique,
    name varchar(255) not null,
    latitude float,
    longitude float,
    address varchar(255),
    deleted int default null,
    primary key(id)
)$
```

4.2.5 storages

A `storages` tábla a különböző raktárak tárolásáért felelős.

```
create table if not exists storages (
    id int not null auto_increment,
    warehouse int not null,
    external_id varchar(255) not null,
    name varchar(255) not null,
    deleted int default null,
    foreign key(warehouse) references warehouses(id),
    primary key(id),
    unique (warehouse,external_id)
) $
```

4.2.6 item_stacks

Az `item_stacks` tábla az éppen tárolt különböző árucikkek tárolásáért felelős.

```
create table if not exists item_stacks (
    storage int not null,
    item_type int not null,
    amount int not null,
    lot varchar(255),
    global_serial_inner int not null default 0,
    global_serial int as (if(global_serial_inner=0,NULL,
        global_serial_inner)) virtual,
    manufacturer_serial varchar(255),
    unique(global_serial),
    foreign key(storage) references storages(id),
    foreign key(item_type) references item_types(id),
    primary key(storage,item_type,lot,global_serial_inner)
) $
```

4.2.7 storage_limits

A `storage_limits` tábla a raktár limiteket tárolásáért felelős.

```
create table if not exists storage_limits (
    storage int not null,
    item_type int not null,
    amount int not null,

    primary key(storage,item_type),
    foreign key(storage) references storages(id),
    foreign key(item_type) references item_types(id)
) $
```

4.2.8 operations

Az `operations` tábla a műveletek tárolásáért felelős.

```
create table if not exists operations (
    id int auto_increment not null,
    external_id varchar(255) not null,
    warehouse int not null,

    name varchar(255),
    type enum ('add','remove'),
    created int not null,
    committed int,

    primary key(id),
    unique(external_id,warehouse),
    foreign key(warehouse) references warehouses(id)
) $
```

4.2.9 operation_items

Az `operation_items` tábla a művelet részek tárolásáért felelős.

```
create table if not exists operation_items (
    operation int not null,
    storage int not null,
    item_type int not null,
    amount int not null,
    lot varchar(255),
    global_serial_inner int not null default 0,
    global_serial int as (if(global_serial_inner=0,NULL,
        global_serial_inner)) virtual,
    manufacturer_serial varchar(255),

    primary key(operation,storage,item_type,lot,global_serial_inner
    ),
    foreign key(operation) references operations(id) on delete
        cascade,
    foreign key(storage) references storages(id),
    foreign key(item_type) references item_types(id)
)$
```

4.2.10 users

A `users` tábla a felhasználók tárolásáért felelős.

```
create table if not exists users (
    id int auto_increment not null,
    external_id varchar(255) not null unique,

    name varchar(255),
    manager int,
    passhash varchar(255) not null,

    primary key(id),
    foreign key(manager) references users(id)
)$
```

4.2.11 authorization

Az `authorization` tábla a helyi engedélyek tárolásáért felelős.

```
create table if not exists authorization (
    user int not null,
    warehouse int not null,

    authorization enum ('view','create_add_operation',
        'create_remove_operation','handle_operation','configure'),

    primary key(user,warehouse,authorization),
    foreign key(user) references users(id),
    foreign key(warehouse) references warehouses(id)
)$
```

4.2.12 global_authorization

A `global_authorization` tábla a rendszer engedélyek tárolásáért felelős.

```
create table if not exists global_authorization (
    user int not null,
    authorization enum (
        "view_warehouses",
        "delete_warehouses",
        "create_warehouses",
        "modify_warehouses",
        "delete_types",
        "create_types",
        "modify_types",
        "view_users",
        "delete_users",
        "create_users",
        "modify_users"),

    primary key(user,authorization),
    foreign key(user) references users(id)
)$
```

4.2.13 authentication_token

Az `authentication_token` tábla a bejelentkezési tokenek tárolásáért felelős.

```
create table if not exists authentication_token (
    token varchar(255) not null,
    user int not null,
    expiration int,
    primary key(token),
    foreign key(user) references users(id)
)$$
```

4.2.14 users_view

A `users_view` a felhasználók adatainak lekérését egyszerűsíti.

```
create or replace view users_view as
    select a.external_id id,a.name `name`,b.external_id manager,b.
        name manager_name,a.passhash passhash from users a
    left join users b on a.manager=b.id$$
```

4.2.15 authorization_view

Az `authorization_view` tábla a helyi és rendszer engedélyek lekérését egyszerűsíti.

```
create or replace view authorization_view as
    select external_id id,authorization,null warehouse from users
        inner join global_authorization on users.id=
            global_authorization.user
    union
    select users.external_id id,authorization,warehouses.
        external_id warehouse from users
        inner join authorization on users.id=authorization.user
        inner join warehouses on warehouses.id=authorization.
            warehouse$$
```

4.2.16 authentication_token_view

Az `authentication_token_view` a bejelentkezési tokenek lekérését egyszerűsíti.

```
create or replace view authentication_token_view as
    select token,external_id user,expiration from
        authentication_token
    inner join users on users.id=authentication_token.user$
```

4.2.17 warehouses_view

A `warehouses_view` tábla a telephelyek lekérését egyszerűsíti.

```
create or replace view warehouses_view as
    select external_id id,name,latitude,longitude,address,deleted
        from warehouses$
```

4.2.18 storages_view

A `storages_view` tábla a raktárak lekérését egyszerűsíti.

```
create or replace view storages_view as
    select warehouses.external_id warehouse,warehouses.name
        warehouse_name,warehouses.address warehouse_address,
        warehouses.deleted warehouse_deleted,storages.external_id id
        ,storages.name,storages.deleted deleted from storages
    inner join warehouses on storages.warehouse=warehouses.id$
```

4.2.19 item_types_view

A `item_types_view` tábla a cikk típusok lekérését egyszerűsíti.

```
create or replace view item_types_view as
    select item_types.external_id id,item_types.name,units.
        external_id unit,units.name unit_name,units.deleted
        unit_deleted,item_types.deleted deleted from item_types
    inner join units on units.id=item_types.unit$
```

4.2.20 units_view

A units_view tábla a mérték egységek lekérését egyszerűsíti.

```
create or replace view units_view as
    select external_id id, name, deleted from units$
```

4.2.21 operations_view

A operations_view tábla a műveletek lekérését egyszerűsíti.

```
create or replace view operations_view as
    select operations.external_id id, warehouses.external_id
        warehouse, warehouses.name warehouse_name, warehouses.address
        warehouse_address, warehouses.deleted warehouse_deleted,
        operations.name, type, created, committed from operations
            inner join warehouses on operations.warehouse=warehouses.
                id$
```

4.2.22 operation_items_view

A operation_items_view tábla a művelet részek lekérését egyszerűsíti.

```
create or replace view operation_items_view as
  select
    operations.external_id operation,
    operations.type `type`,
    operations.committed committed,
    storages.external_id storage,
    storages.name storage_name,
    storages.deleted storage_deleted,
    warehouses.external_id warehouse,
    warehouses.name warehouse_name,
    warehouses.address warehouse_address,
    warehouses.deleted warehouse_deleted,
    item_types.external_id item,
    item_types.name item_name,
    item_types.deleted item_deleted,
    units.external_id unit,
    units.name unit_name,
    units.deleted unit_deleted,
    amount,
    lot,
    global_serial,
    manufacturer_serial
  from operation_items
    inner join operations on operations.id=operation_items.
      operation
    inner join storages on storages.id=operation_items.storage
    inner join item_types on item_types.id=operation_items.
      item_type
    inner join units on units.id=item_types.unit
    left join warehouses on warehouses.id=operations.warehouse$
```

4.2.23 item_stacks_view

A item_stacks_view tábla a tárolt áru lekérését egyszerűsíti.

```

create or replace view item_stacks_view as
  select
    warehouses.external_id warehouse,
    warehouses.name warehouse_name,
    warehouses.address warehouse_address,
    storages.external_id `storage`,
    storages.name storage_name,
    item_types.external_id item,
    item_types.name item_name,
    units.external_id unit,
    units.name unit_name,
    item_stacks.amount amount,
    item_stacks.lot lot,
    item_stacks.global_serial global_serial,
    item_stacks.manufacturer_serial manufacturer_serial,
    (item_stacks.amount-COALESCE(sum(oi.amount),0))
      available_amount
  from item_stacks
    inner join storages on storages.id=item_stacks.storage
    inner join warehouses on storages.warehouse=warehouses.id
    inner join item_types on item_types.id=item_stacks.
      item_type
    inner join units on item_types.unit=units.id
    left join (
      select * from operation_items inner join operations on
        operation_items.operation=operations.id where
        operations.type='remove' and committed is null
    ) oi on (
      oi.storage=item_stacks.storage and
      oi.item_type=item_stacks.item_type and
      oi.lot=item_stacks.lot and
      oi.global_serial_inner=item_stacks.global_serial_inner
    )
  group by
    item_stacks.storage,
    item_stacks.item_type,
    item_stacks.lot,
    item_stacks.global_serial_inner$
```

4.3 Api szerver

4.3.1 Felépítés

Átirányítási réteg

A `.htaccess` -nek köszönhetően minden beérkező kérés az `index.php` fájlba érkezik, amely ezt követően a `src/Routing/Router.php` fájlba továbbítja azokat. A Router.php fájl megvizsgálja a kérések URL -jét és metódusát, majd a query kivételével dekódolja és továbbítja az URL paramétereket a `src/API` mappában található fájlokba.

API kezelő réteg

Az API kérések értelmezése és az engedélyek ellenörzése, illetve az azokra való válasz küldése a `src/API` mappában lévő fájlok feladata. A kérések formátumának ellenörzéséért például a `src/API/APIUtils.php` fájl felelős.

Adatbázis kapcsolat réteg

A `src/Database` mappában található fájlok feladata az adatbázissal való kapcsolat megteremtése és az adatbázis műveletek kezelése.

4.3.2 ApiRouter.php

Amennyiben a kérés metódusa megegyezik a `$method` -al, illetve a kérés URL -je hasonlít a `$pattern` -ben található mintával, a route funkció feladata, hogy lefuttassa a `$function` funkciót az URL -ből kiszedett és dekódolt paraméterekkel.

```
private static function route($method, $pattern, $function): void {
    if($method !== $_SERVER['REQUEST_METHOD'])
        return;
    $url=explode("?",$_SERVER['REQUEST_URI'])[0];
    $url = explode("/",trim($url,"/"));
    $pattern = explode("/",trim($pattern,"/"));
    $params = [];
    if(count($pattern) !== count($url))
        return;

    for($i = 0; $i < count($pattern); $i++)
        if($pattern[$i]==":")
            $params []= urldecode($url[$i]);
        else if($url[$i] != $pattern[$i])
            return;
    $function(...$params);
}
```

Az API -ra érkező kérés hatására a `handle` nevű funkció hívódik meg, amely a `route` segítségével átirányítja a dekódolt kéréseket az api kezelő rétegbe.

```
class ApiRouter {
    public static function handle() { ... }
    ...
}
```

4.3.3 APIUtils.php

A validate funkció feladata megvizsgálni, hogy a `$data` értéke a `$template` szerint van-e formázva. Amennyiben a `$template` egy szöveg, akkor a funkció a `$data` típusát ellenőrzi. Ilyen értékek például:

- "string"
- "integer"
- "double"

(Ha a `$template` szerint tört szám kell, akkoraz egész szám is elfogadott, mivel egyszerűen átalakítható.)

Amennyiben a `$template` egy lista, ellenőrzi, hogy a `$data` is lista-e. Ezt követően a `$template` az első elemét használja mintaként és rekurzívan ellenőrzi a `$data` lista elemeit. Ilyen értékek például:

[**"string"**] Olyan listákat fogad el amiben csak szövegek vannak.

[**"integer"**] Olyan listákat fogad el amiben csak egészek vannak.

[**"double"**] Olyan listákat fogad el amiben csak törtszám listák vannak.

Abban az esetben, ha a `$template` egy asszociatív lista, a validate funkció ellenőrzi, hogy a `$data` is asszociatív lista e. Ezt követően a `$template`, saját értékeit mintaként használva, rekurzívan ellenőrzi a `$data` értékeit. Amennyiben a `$template` kulcsa "?" -el végződik, nincs szükség rá, hogy a `$data` -nak is meglegyen. Ilyen értékek például:

[**"key"=>"integer"**] Ez elfogadja a ["key"=>10] értéket.

[**"key?"=>"string"**] Ez elfogadja a ["key"=>"szöveg"] és [] értékeket.

```
static function validate(array|string $template,mixed $data) {
    $datatype=gettype($data);
    if(gettype($template)==="string") {
        if(!self::isCompatible($template,$datatype))
            Response::badRequest();
    }else{
        if($datatype!="array")
            Response::badRequest();
        if(isset($template[0])){
            foreach($data as $d) {
                self::validate($template[0],$d);
            }
        }else{
            foreach($template as $key=>$templ) {
                if(str_ends_with($key,"?")){
                    $d=$data[substr($key,0,strlen($key)-1)]??null;
                    if($d!==null)
                        self::validate($templ,$d);
                }else
                    self::validate($templ,$data[$key]??null);
            }
        }
    }
}
```

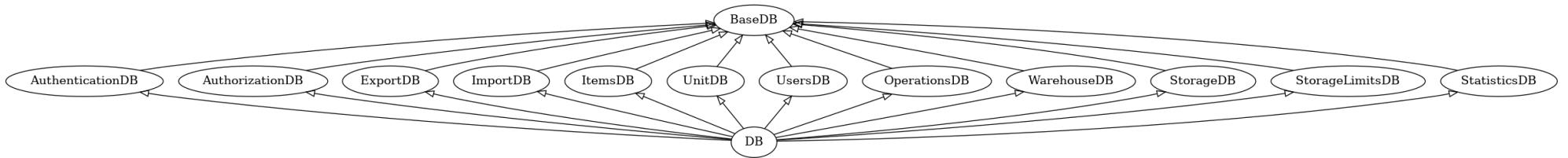


Figure 4.1: Osztálydiagramm

4.3.4 DB.php

A DB osztály felelős az adatbázishoz való kapcsolódásért. A legtöbb metódus külön `trait`-ekben van implementálva, melyeket ez az osztály használ. Ez a megoldás nagyban javítja az olvashatóságat, mivel a hasonló feladatokat ellátó funkciók ugyanabban a fájlban vannak és nem minden a `DB.php`-ban.

4.3.5 BaseTrait.php

Sok hasznos SQL lekérdezéssel foglalkozó metódus van implementálva a `BaseTrait`-ben, így az összes többi `trait` használja a `BaseTrait`-et ezeknek a bizonyos funkcióknak az eléréséhez.

4.3.6 Logger.php

A `Logger` osztály az eseménynapló kezelésével foglalkozik.

```
class Logger {
    function __construct($src_ip,$src_port,$useragent,$http,$method
        ,$url,$req_body) { ... }
    function __destruct() { ... }
    public static function set_logger(self $l):void { ... }
    public static function set_userid(string|null $userid):void {
        ...
    }
    public static function set_response(int $code,string $body):
        void { ... }
    public static function log(string $message):void { ... }
    public static function error($e):void { ... }
}
```

4.3.7 Settings.php

A `Settings` osztály a `config.json` olvasásáért felelős.

```
class Settings {
    public readonly DBConnection $db_connection;
    public readonly LogSettings $success_log;
    public readonly LogSettings $bad_log;
    public readonly LogSettings $rejected_log;
    public readonly LogSettings $error_log;
    public readonly string|null $test_token;
    public readonly int $token_length;
    string|null $test_token,int $token_length) { ... }
    public static function getSettings():Settings { ... }
}
```

4.3.8 Authentication.php

A `Authentication` osztály a hiteletisért felelős.

```
class Authentication {
    public static function getToken():string|null { ... }
    public static function isTestUser():bool { ... }
    public static function getCurrentUserId():string|null { ... }
    public static function isValidPassword(string $password):bool {
        ...
    }
    public static function verifyPassword(string $user_id,string
        $password):bool { ... }
    public static function authenticate(string $user_id,string
        $password):string|null { ... }
    public static function createHash(string $password): string {
        ...
    }
}
```

4.4 Web frontend

4.4.1 Áttekintés

A webes frontend html része több fájlba van tagolva az olvashatóság érdekében. A html nagy része sablon amit a javascript használ az oldal megjelenítéséhez. Ez lehetővé teszi az újratöltés mentes navigálást.

4.4.2 Építés

A frontend építéséhez a Linux alapú rendszeren `build` vagy Windowson `build.bat` fájlt kell futtatni. Ez kétrehozza az `index.html` fájlt a széttagolt html fájlokból.

4.4.3 Template osztály

A `Template` osztály segíti a html minták kezelését. A htmlben osztályokkal vannak megjelölve azok az elemek amiket JavaScript kód módosít. A `get` funkció ezek az elemek lekérését segíti. A `getNode` funkció visszaadja az új template gyökér elemét.

```
class Template {
    template;
    constructor(template) { ... }
    get(classname) { ... }
    getNode() { ... }
}
```

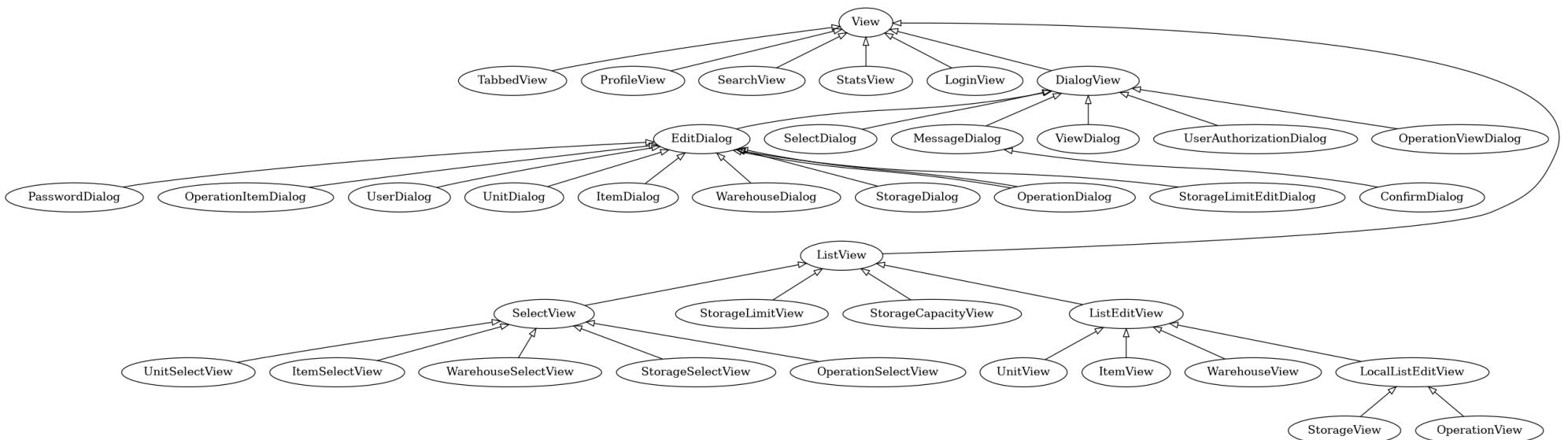


Figure 4.2: Osztálydiagramm

4.4.4 View osztály

A `View` osztály a felhasználói felület részek szülőosztálya. Ebben az osztályban az `initTemplate` metódus egyszer hívódik meg és a feladata az, hogy inicializálja a felhasználói felület részt.

```
class View {
    view_template=null;
    template_id;
    constructor(template_id) { ... }
    getNode() { ... }
    initTemplate() {}
}
```

4.4.5 ListView osztály

A `ListView` osztály a listák szülőosztálya. Ebben a listában lehet lapozni, keresni és újratölteni. A `load` funkció feladata, hogy a megadott paraméterekre (keresési szöveg, archivált-e, kezdő pozíció és mennyiség) vissza adjon egy `Promise` objektumot, amely később vissza adja majd a betöltött értékeket. A töltés animáció egész addig látható a felületen, amíg a Promise nem ad vissza értéket vagy hibát. A `createItemCard` funkció feladata, hogy a betöltött értékeket vizualizálja.

```
class ListView extends View {
    offset=0;
    content;
    spinner;
    query;
    prev;
    next;
    archived;
    reloading=false;
    constructor() { ... }
    initTemplate() { ... }
    reload() { ... }
    load(q,archived,offset,length) {}
    createItemCard(v) {}
}
```

4.4.6 ListView osztály

A `ListEditView` osztály a `ListView` osztályt szerkesztési menüvel egészíti ki. A `createEditView` funkció feladata az, hogy létre hozza a szerkesztési menüt. A `item` a szerkeszteni kívánt érték. Ez `null` ha újat akarunk létrehozni. Az `item_callback` funkciót kell meghívnia a menünek a mentéshez. Ennek a paraméterei az eredeti érték és az új érték.

A `save` funkció feladata az új vagy módosított érték mentése.

```
class ListEditView extends ListView {
    initTemplate() { ... }
    createItemCard(v) { ... }
    createItemNode(item) {}
    createEditView(item, item_callback) {}
    delete(item) {}
    save(original, modified) {}
    initEditDelete(tmp, v) { ... }
}
```

4.4.7 SelectView osztály

A `SelectView` osztály a kiválasztási listák szülőosztálya. Ez az osztály a `ListView` osztályt egészíti ki és ezért ebben a listában is lehet lapozni, keresni és újratölteni.

```
class SelectView extends ListView {
    selected=null;
    selectedNode=null;
    getSelected() { ... }
    initTemplate() { ... }
    createItemCard(v) { ... }
}
```

4.4.8 DialogView osztály

A `DialogView` osztály a felugró menük szülőosztálya. A `hasOk` funkció igazat ad, amennyiben ennek a menünek van OK gombja. Ebben az esetben implementálni kell az `ok` funkciót. Ez a funkció az OK gomb lenyomásakor hívódik meg és egy `Promise` objektumot ad vissza, majd a töltés animáció egész addig látható lesz, amíg ez az objektum kész nem lesz.

```
class DialogView extends View {
    ok_button=null;
    constructor(template_id) { ... }
    ok() {}
    hasOk() { ... }
}
```

4.4.9 EditDialog osztály

A `EditDialog` osztály a szerkesztési menük szülőosztálya. A `getItem` funkció visszaadja a módosított értéket vagy hibát ha rosszul van kitöltve a menü.

```
class EditDialog extends DialogView {
    item_callback;
    item;
    constructor(template_id,item_callback,item) { ... }
    validate() { ... }
    getItem() {}
    ok() { ... }
    hasOk() { ... }
}
```

4.5 API könyvtár

4.5.1 Áttekintés

Az API könyvtárat a mobil és az asztali alkalmazás használja ahoz, hogy kommunikáljon az API szerverrel. A könyvtárban az **API** osztály implementálja az összes funkciót amikkel tudunk kommunikálni az API-val. A használatáról található egy részletesebb leírás a saját dokumentációjában.

4.5.2 HTTPConnector osztály

A **HTTPConnector** egy absztrakt osztály, amelynek feladata, hogy az adott paraméterek alapján létrehozzon egy kérést, majd vissza adja annak eredményét. Annak érdekében, hogy az osztályt használó programok hatékonyabban irányíthassák a kéréseket a könyvtár használata közben, ez az osztály nincs beépítve.

4.6 Asztali kliens

4.6.1 Bevezetés

Az asztali kliens az API könyvtár segítségével kommunikál az API szerverrel. A kliens a JavaFX felhasználói felület platformot használja. A fejlesztéshez a Netbeans IDE használható.

4.6.2 Worker osztály

A `Worker` osztály a háttérben futó folyamatok kezeléséért felelős.

4.6.3 EasyInventoryDesktop osztály

A program az `EasyInventoryDesktop` osztályban kezdődik, ahol a kezdő metódus betölti a nyelvfájlokat, beállítja az ablakot és elindítja a bejelentkezési felület létrehozását. Emellett ez az osztály állítja be az eseménykezelőket és hozza létre a bejelentkezés utáni felületet is.

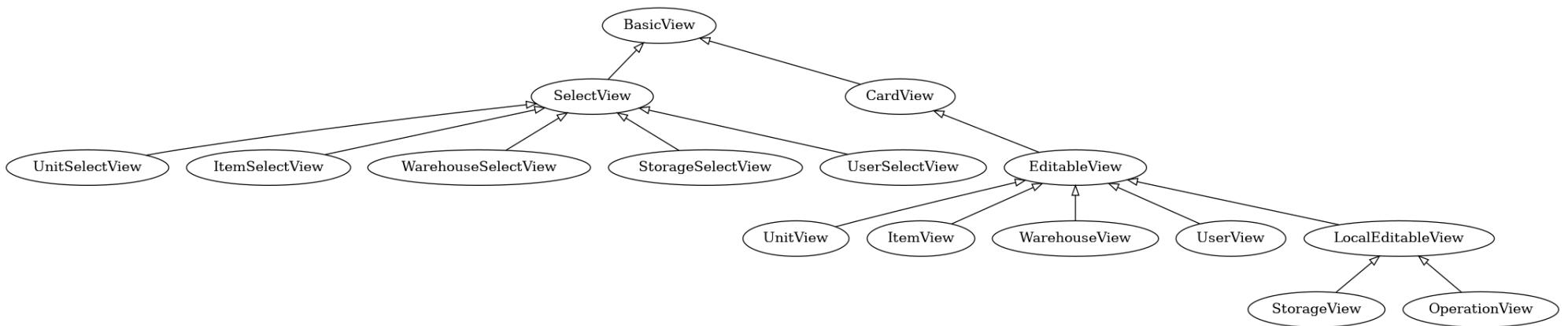


Figure 4.3: Osztálydiagramm

4.6.4 BasicView osztály

A `BasicView` absztrakt osztály a különböző listázási felületek szülőosztálya. Ez kezeli a listázáshoz tartozó újratöltés, keresés és pagináció menüelemeket. A `reload` funkció betölti a lista elemeit a megadott paraméterek alapján (keresési szöveg, kezdő pozíció, mennyiség és archivált-e). Ezt a funkciót a `Worker` szála hívja meg. A `createEntry` funkció feladata, hogy vizualizálja a betöltött adatokat.

4.6.5 SelectView osztály

A `SelectView` absztrakt osztály a különböző kiválasztási felületek szülőosztálya. Ez az osztály implementálja a `createEntry` funkciót, amely a `showInfo` meghívásáért felel, amely vizualizálja a betöltött adatot. A `showInfo`-nak nem kell kiválasztás gombot hozzáadnia, mivel azt a `createEntry` megteszí helyette.

4.6.6 CardView osztály

A `CardView` absztrakt osztály olyan listázási felületek szülőosztálya amelyek káryákban jelenítik meg az adatokat. A `createActionButtons` funkció feladata a akció gombot létrehozása. A `getArchivalUnixtime` funkció feladata az archiválás idejének visszaadása, `Long.MAX_VALUE` ha nem archivált. A `showInfo` funkció feladata, hogy vizualizálja az adatot a kártyán belül.

4.6.7 EditableView osztály

A `EditableView` absztrakt osztály a `CardView` osztályt szerkesztési gombokkal egészíti ki. Ez implementálja a `createActionButtons` funkciót szerkesztés és törlés gomb létrehozásával. A `showDialog` funkció feladata a szerkesztési és létrehozási dialógus létrehozása. A `delete` funkció feladata az adat törlése.

4.6.8 LocalEditableView osztály

A `LocalEditableView` absztrakt osztály a `EditableView` osztályt telephely kiválasztási menüvel egészíti ki.

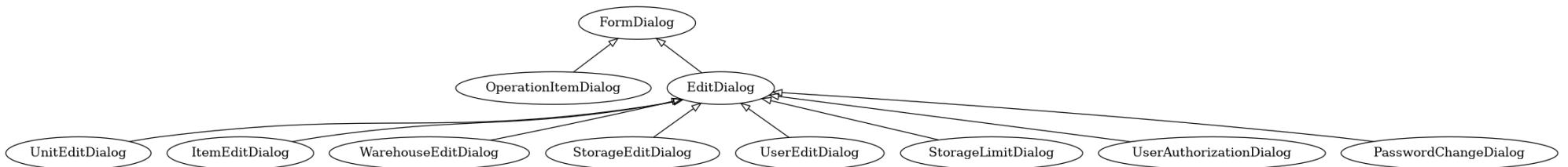


Figure 4.4: Osztálydiagramm

4.6.9 FormDialog osztály

A `FormDialog` absztrakt osztály a bemeneti felügrő ablakok szülőosztálya.

A `createEditFields` funkció a bemenetek lérhehozásáért felelős. Az `applyEdits` funkció a bemenet értelmezéséért felelős.

4.6.10 EditDialog osztály

Az `EditDialog` absztrakt osztály a `FormDialog` osztályt mentéssel egészíti ki.

4.7 Mobil kliens

4.7.1 Áttekintés

A mobil alkalmazás forrás fájljai a `EasyInventoryMobile` map-pában találhatóak, fejlesztésre pedig az Android Studió használható. A nyelvfájlokat a `langconvert.py` python program generálja. Az ikonok megegyeznek a webes frontend ikonjaival, viszont importálva lettek, így ikon-módosítás esetén érdemes újra importálni.

4.7.2 MainActivity osztály

A program a `MainActivity` osztályban kezdődik, ami megjeleníti a be-jelentkezési felületet, illetve amennyiben még nincs, engedélyt kér a hálózat használatához.

```
public class MainActivity extends AppCompatActivity {
    public static easyinventoryapi.API api;
    public static String user_id, user_name;
    public static boolean isStopped() { ... }
    @Override
    protected void onCreate(Bundle savedInstanceState) { ... }
}
```

4.7.3 LoggedInActivity osztály

A `LoggedInActivity` osztály megjeleníti a bejelentkezett felületet. A különböző fülek kódjai a saját `Fragment` osztájaikban vannak.

```
public class LoggedInActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) { ... }
    public void setToolbarController(ToolbarController t) { ... }
    @Override
    public boolean onSupportNavigateUp() { ... }
    public interface ToolbarController { ... }
}
```

4.7.4 Worker osztály

A Worker osztály a háttérben futó folyamatok kezeléséért felelős.

```
public class Worker extends Thread {  
    public static final Worker GLOBAL=new Worker();  
    @Override  
    public void run() { ... }  
    public void addTask(Runnable task) { ... }  
}
```

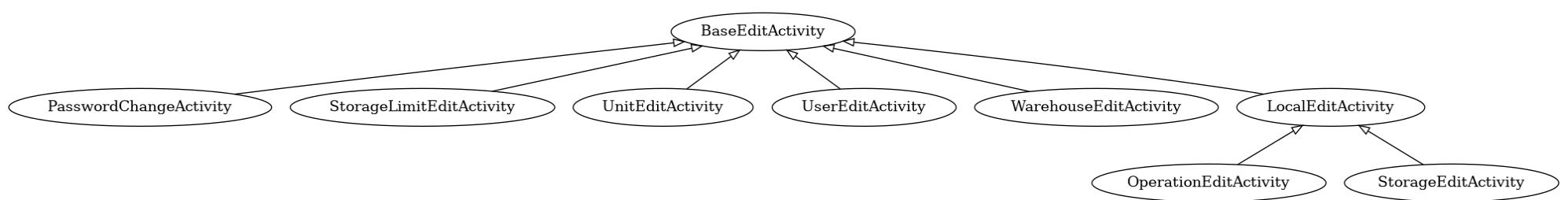


Figure 4.5: Osztálydiagramm

4.7.5 BaseEditActivity osztály

A `BaseEditActivity` absztrakt osztály a szerkesztési vagy hozzáadási tevékenységeknek a szülőosztálya. A `initUI` metódus feladata a felhasználói felület inicializálása. A `getItem` funkció feladata a bemeneti adat értelmezése. A `save` metódus feladata az adat mentése. A `readContext` funkció feladata a módosítandó adat kiolvasása a tevékenység adatokból.

```
public abstract class BaseEditActivity<T> extends AppCompatActivity
{
    public BaseEditActivity(int layout) { ... }
    protected abstract void initUI(@Nullable T item);
    protected abstract @NonNull T getItem();
    protected abstract void save(@Nullable T original, @NonNull T
        item) throws FormattedException;
    protected abstract T readContext(Intent i);
    protected void readExtraContext(Intent i) {}
    @Override
    protected void onCreate(Bundle savedInstanceState) { ... }
    @Override
    public boolean onSupportNavigateUp() { ... }
}
```

4.7.6 LocalEditActivity osztály

A `LocalEditActivity` absztrakt osztály a `BaseEditActivity` osztályt telephely bekéréssel egészíti ki. Ilyen tevékenység indításánál meg kell adni a kiválasztott telephely azonosítóját a `warehouse_id` kulcsal.

```
public abstract class LocalEditActivity<T> extends BaseEditActivity
<T> {
    public LocalEditActivity(int layout) { ... }
    @Override
    protected void readExtraContext(Intent i) { ... }
    public String getSelectedWarehouseId() { ... }
}
```

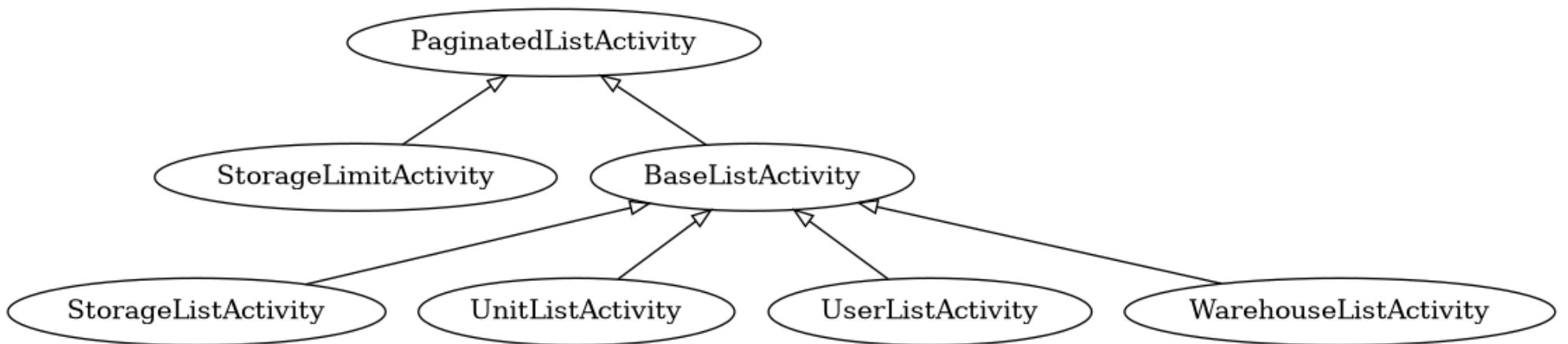


Figure 4.6: Osztálydiagramm

4.7.7 PaginatedListActivity osztály

A `PaginatedListActivity` absztrakt osztály olyan tevékenységek szülőosztálya, amelyek lapozható listát mutatnak. A `load` funkció feladata, hogy a megadott paraméterek (keresési szüveg, kezdőpozíció, mennyiség, archivált-e) alapján betöltsse az adatokat.

```
public abstract class PaginatedListActivity<T> extends
    AppCompatActivity {
    public static final int PAGE_SIZE=20;
    protected int offset=0;
    protected SearchView search;
    protected SelectableArrayAdapter<T> adapter;
    protected SwipeRefreshLayout sw;
    protected PaginatedListActivity(int activity, Function<Activity,
        SelectableArrayAdapter<T>> adapter_factory) { ... }
    protected void reload() { ... }
    protected abstract T[] load(String query, int offset, int length,
        boolean archived) throws FormattedException;
    protected void readExtraContext(Intent i) {}
    @Override
    protected void onCreate(Bundle savedInstanceState) { ... }
    @Override
    public boolean onSupportNavigateUp() { ... }
}
```

4.7.8 BaseListActivity osztály

A `BaseListActivity` absztrakt osztály a kiválasztási felületek szülőosztálya. A `getNullDisplay` funkció feladata, hogy amennyiben a `load` funkció által visszaadott értékek között `null` érték is található, kicserélje azt a funkció visszatérési értékére.

```
public abstract class BaseListActivity<T extends ToJSON> extends
    PaginatedListActivity<T> {
    protected BaseListActivity(Function<Activity,
        SelectableArrayAdapter<T>> adapter_factory) { ... }
    protected abstract T getNullDisplay();
    @Override
    protected void reload() { ... }
    protected abstract T[] load(String query, int offset, int length,
        boolean archived) throws FormattedException;
    @Override
    protected void onCreate(Bundle savedInstanceState) { ... }
}
```

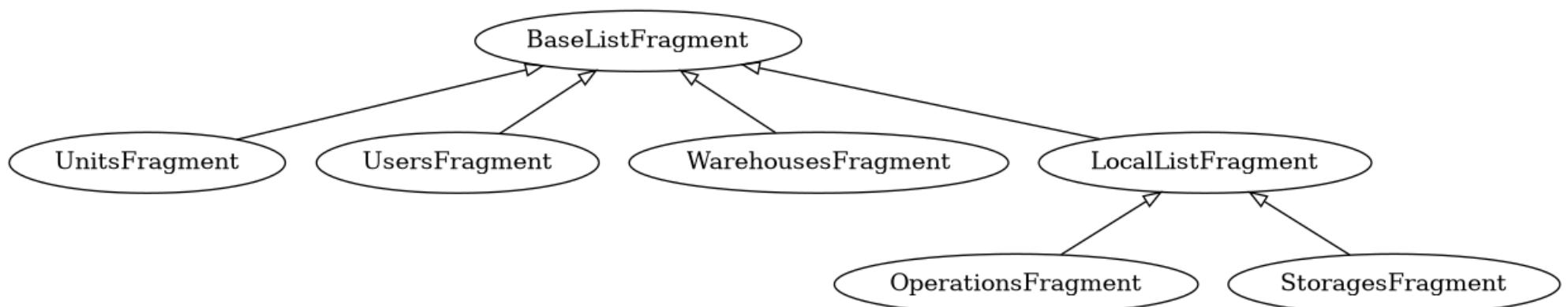


Figure 4.7: Osztálydiagramm

4.7.9 BaseListFragment osztály

A `BaseListFragment` absztrakt osztály a lista `Fragment`-ek szülőosztálya, melynek feladata a lapozás, a keresés, az újratöltés, a törlés és a szerkesztés indítása. A `PaginatedListActivity` osztályhoz hasonlóan itt is megtalálható a `load` funkció. Az adat törlése a `delete` metódus feladata.

```
public abstract class BaseListFragment<T extends ToJSON> extends Fragment implements LoggedInActivity.ToolbarController {
    protected abstract T[] load(String query, int offset, int length,
        boolean archived) throws FormattedException;
    protected abstract void delete(String id) throws FormattedException;
    protected void reload() { ... }
    protected void addExtraContext(Intent i) {}
    protected boolean canAdd() { ... }
    protected void initRoot(View root, SelectableArrayAdapter<T>
        adapter, Function<T, String> id_getter, Class<?> edit_activity
    ) { ... }
    @Override
    public void controlToolbar(TextView title, ImageButton
        warehouse_button, SearchView search_bar) { ... }
    protected void startActivity(Class<?> activity) { ... }
    protected void startActivityForSelected(Class<?> activity) {
        ... }
}
```

4.7.10 LocalListFragment osztály

A LocalListFragment absztrakt osztály a BaseListFragment osztályt telephely bekéréssel egészíti ki.

```
public abstract class LocalListFragment<T extends ToJSON> extends
BaseListFragment<T> {
    @Override
    protected void addExtraContext(Intent i) { ... }
    @Override
    protected boolean canAdd() { ... }
    @Override
    protected void initRoot(View root, SelectableArrayAdapter<T>
        adapter, Function<T, String> id_getter, Class<?>
        edit_activity) { ... }
    @Override
    public void controlToolbar(TextView title, ImageButton
        warehouse_button, SearchView search_bar) { ... }
    public @Nullable Warehouse getSelectedWarehouse() { ... }
}
```

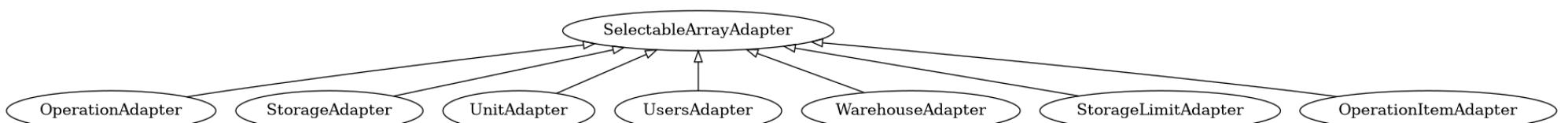


Figure 4.8: Osztálydiagramm

4.7.11 SelectableArrayAdapter osztály

A `SelectableArrayAdapter` absztrakt osztály az `ArrayAdapter` osztály implementálását segíti kijelölhető listákra.

4.8 Dokumentáció

4.8.1 Áttekintés

A projekt dokumentációja L^AT_EX dokumentum leíró nyelvet használja. A forrás fájlok a `doc/src` mappában találhatók. A legtöbb stílus elem és a csomagok importálása a `shared.tex` fájlból van beállítva.

4.8.2 Felhasználói dokumentáció

A felhasználói dokumentáció a `user` mappában található. A dokumentum által használt ikonok a `user/resources/icon` mappában találhatóak. Az ikonok megváltoztatása esetén az itt található ikonokat is frissíteni kell. A `user/resources/web`, `user/resources/mobile` és `user/resources/desktop` mappákban található képeket a programok E2E tesztjei készítik. A programok kinézetének változtatásakor az itteni képeket érdemes frissíteni.

4.8.3 Kód dokumentáció

A kód dokumentáció a `code` mappában találhatóak. Az dokumentum által használt kód részek a `code/generated` mappában találhatóak. Ezeket a kód részleteket az építő program autómatikusan generálja.

4.8.4 Teszt dokumentáció

A teszt dokumentáció a `test` mappában, a dokumentum által használt kód részletek pedig a `test/generated` mappában találhatóak. Ezeket a kód részleteket az építő program autómatikusan generálja.

4.8.5 Kód beillesztő

A kód beillesztő program a `codeextractor.py` python program.

4.8.6 Építés

A dokumentáció építéséhez a `compile` fájlt kell futtatni. Ez generált pdf fájlokat az `out` mappába helyezi.

Teszt Dokumentáció

Fontos!

A tesztelő rendszerek teljesen átírják az adatbázist.

5.1 API szerver

5.1.1 Áttekintés

Az API szerver teszteléséért az `apitest` mappában található python fájlok felelősek. A tesztek lefuttatása előtt fel kell állítani a teszt API szervert. Első lépésként az API szerver konfigurációs fájljában (`config.json`) kell beállítani a teszt tokent, majd ugyan ezt el kell végezni a teszt konfigurációs fájlban is. A sikeres beállítás után a `main.py` futtatásával indíthatók el a tesztek.

5.1.2 E2E tesztek

Az E2E tesztek az API-t hívják és ellenőrzik a válasz kódot és adatot. A tesztelés megkezdése előtt az rendszer visszatölt egy megadott biztonsági mentést, ami tartalmazza a tesztadatokat.

5.2 Web frontend

5.2.1 Áttekintés

A webes frontend tesztelésére csupán E2E autómata teszteket használunk, mivel ezek kellően letesztelek azt, így nincs szükség további tesztekre.

5.2.2 E2E tesztek

Az E2E tesztekhez python alapú playwright keretrendszert használunk. Az E2E tesztek minden műveletet végrehajtanak kattintás és gépelés szimulációval. Helyes és hibás bemeneti adatok egyaránt tesztelve vannak, illetve a tesztek egy része képernyő képeket készít a program működéséről, amiket a dokumentációhoz használunk fel.

5.3 Asztali alkalmazás

5.3.1 Áttekintés

Az asztali alkalmazás a TestNG tesztelési keretrendszert használja.

5.3.2 E2E tesztek

Az asztali alkalmazás tesztelése olyan E2E teszteket történik, amelyek felhasználói interakciókat szimulálnak és megjelenített szövegeket ellenőriznek. A felhasználói felület elemeinek tesztelhetősége érdekében az elemek CSS osztályokkal vannak ellátva. A tesztek egy része képernyőképeket készít a dokumentációhoz.

5.3.3 NodeMatcher osztály

A `NodeMatcher` osztály feladata a felhasználói felület elemeinek kiválasztása és az interakció szimulálása a kiválasztott elemekkel.

```
public class NodeMatcher {  
    public static NodeMatcher allWindowRoots() { ... }  
    public NodeMatcher descendants() { ... }  
    public NodeMatcher withClass(String classname) { ... }  
    public NodeMatcher labels(String text) { ... }  
    public NodeMatcher textFields() { ... }  
    public NodeMatcher tabPanes() { ... }  
    public NodeMatcher buttons() { ... }  
    public NodeMatcher with(Function<NodeMatcher,NodeMatcher> m) {  
        ... }  
    public static void sync(Runnable r) { ... }  
    public Node get() { ... }  
    public int count() { ... }  
    public void replaceText(String text) { ... }  
    public void replaceNumber(int value) { ... }  
    public void click() { ... }  
    public void clickASync() { ... }  
    public void selectTab(int index) { ... }  
    public void exists() { ... }  
    public void doesNotExists() { ... }  
    public void acceptDialog() { ... }  
    public void closeDialogs() { ... }  
    public void print() { ... }  
    public void screenshotWindow(String out) { ... }  
}
```

allWindowRoots

A `allWindowRoots` funkció létrehoz egy `NodeMatcher` objektumot amiben minden ablak gyökér eleme ki van választva.

descendants

A `descendants` metódus létrehoz egy `NodeMatcher` objektumot amiben azok az elemek vannak kijelölve amik legalább egy kijelölt elem leszármazottja.

with

A `with` metódus bekér egy lambda függvényt, amellyel egy `NodeMatcher` objektumot egy másik `NodeMatcher` objektummá alakít át. Erre a folyamatra azért van szükség, hogy a metódus le tudja szűrni a kijelölt elemeket és elérni, hogy csak olyan kiválasztást adjon vissza, amelyben legalább egy elem található. A függvény bemenete minden esetben egy olyan kijelölés, amiben megtalálható az az elem, amit éppen szűr. A leszűrés végén a metódus a kijelölést egy új `NodeMatcher` objektum formájában kiadja. Ez például arra használható ha csak azokat az elemeket akarjuk amikben van egy bizonyos címke.

withClass

A `withClass` metódus egy olyan `NodeMatcher` objektumot hoz létre, amelyben csak azok az elemek vannak kiválasztva, amelyek el vannak látva a megadott CSS stílussal.

click

A `click` metódus kattintást szimulál a kiválasztott gombon. Ha nem gomb van kiválasztva vagy nem egy elem van kiválasztva akkor hibát dob.

replaceText

A `replaceText` metódus szöveg beírását szimulálja a kiválasztott bemeneti mezőn. Ha nem bemeneti mező van kiválasztva vagy nem egy elem van kiválasztva akkor hibát dob.

screenshotWindow

A `screenshotWindow` metódus a kijelölt elem ablakáról készít egy képernyőképet és a megadott névvel elmenti. Ha egy elem van kiválasztva akkor hibát dob.

5.4 Mobil alkalmazás

5.4.1 Áttekintés

A mobil alkalmazás tesztelése J4Unit teszt keretrendszerrel történik.

5.4.2 E2E tesztek

A mobil alkalmazás tesztelése olyan felhasználói felület tesztekkel történik, amellyek kattintásokat és szövegek beírását szimulálják. Ezek a tesztek ugyanazt tesztelik, amit az asztali alkalmazás E2E tesztjei, annyi különbösséggel, hogy ezek az **Espresso** UI tesztelési keretrendszeret használják a felhasználói interakciók szimulálására. A tesztekkel helyes és hibás be-meneti adatok is egyaránt tesztelve vannak, illetve a tesztek egy része képernyőképeket készít, amelyeket a dokumentációhoz használunk fel.

Felhasználói Dokumentáció

6.1 Bevezetés

Az alkalmazás célja, hogy lehetővé tegye cégek számára, hogy bármiféle képesítés vagy különösebb hozzáértés nélkül is egyszerűen tudják kezelní a legkomplexebb nyilvántartásokat is. A program lehetővé fogja tenni, hogy különböző jogköröket csatoljunk az egyes felhasználókhöz, melyek befolyásolják, hogy a nyilvántartás mely részeit tekintheti meg és milyen műveleteket hajthat végre a programon belül az adott illető. A rendszer képes lesz nyilvántartani a telephelyeket, raktárakat és cikkeket, miközben figyelembe veszi a különböző egységek és tárolóhelyek közötti kapcsolatok kezelését is. Mindezek mellett, egyik fő szempontunk, hogy a rendszer a lehető leg helytakarékosabb módon rendezze el az adott cikkeket a raktárokon és telephelyeken belül, figyelembe véve azt is, hogy ne kerülhessenek egymás mellé olyan anyagok, amelyek reakcióba lépése károsodásokhoz vezethet. Az alábbiakban részletesen ismertetjük a program főbb funkcióit és kezelési lehetőségeit.

6.1.1 Fogalmak

Telephelyek

A telephelyek különböző telkek vagy raktárépületek, amelyek területén különböző raktárak találhatóak.

Raktárak

Raktárak a legkissem tároló egység amikben vannak az árucikkek nyilvántartva. Ezeknek meg lehet adni, hogy cikkenként mennyi fér beléjük.

Műveletek

A műveletek segítségével tudunk cikkeket hozzáadni vagy éppen eltávolítani a raktárakból. Mint arra a korábbi mondat is utalt, két féle művelet létezik: a hozzáadási és az eltávolítási.

Hozzáadási művelet

A hozzáadási művelet használatával különböző cikkeket tudunk felvinni egy adott raktár nyilvántartásába. Létrehozásakor lehetőségünk van különböző követelmények megadására is, melyekkel szabályozhatjuk, hogy hová kerülnek a hozzáadásra szánt cikkek.

Eltávolítási művelet

Az eltávolítási művelet segítségével lehetőségünk van cikkek kivételére a hozzájuk tartozó raktárból. Létrehozásakor a hozzáadási művelethez hasonló képpen itt is lehetőségünk van szabályozni, hogy melyik raktárból és melyik cikkek kerüljenek eltávolításra, amit meghatározhatjuk lot-, vagy akár sorozatszám megadásával is.

Lot számok

A lot számok különböző cikk csoportok megkülönböztetésére szolgáló kódok, amelyek nagyobb nyilvántartások kezelésének esetén nélkülözhetetlenek.

Globális sorozat szám

A globális sorozatszám egy olyan szám, amelynek az egész rendszeren belül egyedinek kell lennie, még az azonos árucikkek között is.

Gyártói sorozat szám

Minden hozzáadott árucikkhez hozzárendelhetünk egy gyártói sorozatszámot. Ezeknek a számoknak nem feltétlenül kell egyedieknek lenniük, mivel ha több gyártó állítja elő az adott árucikket akkor a sorozatszámaik akár ütközhetnek is.

6.2 Telepítés

6.2.1 Server telepítés

Linux mint

A szerver telepítéséhez a futtatni kell a telepítő programot.

```
chmod +x install.sh && ./install.sh
```

Ez a parancs telepíti szükséges csomagokat és bemásolja a frontendet és a szervert a `/var/www/html/` mappába. Ezután ott be tudjuk állítani a szervert az `api/config.json` fájlban.

Más linux alapú rendszerek

A szerver működéséhez a következő programok szükségesek:

- php
- php-mysqli
- apache
- libapache2-mod-php
- mariadb-server

A szerver telepítéséhez az `api` és a `frontend/assets` mappát és a `frontend/index.html` fájlt át kell másolni az apache web mappába. A mysql szerveren létre kell hozni egy felhasználót és be kell állítani. Az apache szerveren engedélyezni kell a `.htaccess` fájlokat és be kell kapcsolni a `rewrite` és a `PHP` modulokat.

Windows

A szerver Windowsra való telepítéséhez először a XAMPP-ot kell telepítenünk. Utána az `api` és a `frontend/assets` mappát és a `frontend/index.html` fájlt át kell másolni a `htdocs` mappába.

6.2.2 Asztali kliens telepítés

Az asztali kliens fájljai a `EasyInventoryDesktop/dist/bundles/EasyInventoryDesktop` mappában találhatóak. Ezt a mappát akárhova lehet másolni és nem igényel telepítőt, mert mindegyik szükséges dll benne van a mappában. A futtatáshoz a benne található exe fájlt lehet futtatni.

6.2.3 Mobil kliens telepítés

A mobil alkalmazás telepítéséhez Android 14 szükséges.

6.3 Adminisztráció

6.3.1 Arhivált adatok törlése

Fontos!

Az arhivált adatok törlése visszafordíthatatlan! Ezek az adatok adják a statisztikákat. Régi adatok törlésével az akkori statisztika is törlődik.

Az arhivált adatok törlése hasznos lehet ha az adatbázis mérete már túl nagy.
Az összes adat törlésére az alábbi parancsot kell futtatni

```
php archive delete all
```

Általában nem akarjuk az összes arhivált adatot törölni. Ehez meg kell adni hány évnyi, heti vagy napi adatot akarunk megtartani az időegység jelével.

d nap

w hét

y év

Például: ez a parancs kitöli az összes arhivált adatot ami 5 évnél régebben lett törölve.

```
php archive delete 5y
```

6.3.2 Eseménynapló kezelése

Az eseménynapló sok hasznos információt tárol a szerver működéséről. Ezekhez az információk segíthetnek a hibás beállítások javításán. Az eseménynaplót a parancssoron keresztül kezelhetjük.

Eseménynapló megtekintése

Az eseménynapló megjelenítésére van a `log show` parancs. Ha nem adunk meg semmilyen kapcsolót akkor az összes adatot rövid formában írja ki.

```
php log show
```

-u, -user kapcsoló

Ezzel megadhatjuk a felhasználói azonosítót amihez tartozó eseményeket akarjuk megtekinteni.

-e, -error kapcsoló

Szerverhibát okozó kéréseketre szűri le az eseményeket. Ezek lehetnek adatbázis kapcsolódási hibák.

-r, -rejected kapcsoló

Elutasított kérésekre szűri le az eseményeket.

-b, -bad kapcsoló

Hibás kérésekre szűri le az eseményeket.

-s, -success kapcsoló

Sikeres kérésekre szűri le az eseményeket.

-v, -verbose kapcsoló

Az eredményeket hosszú formában jeleníti meg.

Pédák

Megmutatja az összes kérést amit az `admin` felhasználó végzett

```
php log -u admin show
```

Megmutatja az összes szerverhibát és elutasított kérést

```
php log -er show
```

Megmutatja az összes szerverhibát részletesen

```
php log -ev show
```

Egy esemény megtekintése

Ha csak egy kérést akarunk megtekinteni akkor a kérés sorszámának megadásával tudjuk. Ehez a kéréshez nincsenek kapcsolók.

```
php log show 100
```

Eseménynapló törlése

Ez a parancs törli a teljes eseménynapló tartalmát:

```
php log clear
```

Általában nem akarjuk az összes eseményt törölni. Ehez meg kell adni hány évnyi, heti vagy napi adatot akarunk megtartani az időegység jelével.

d nap

w hét

y év

Például: ez a parancs kitöli az összes eseményt ami 5 napnál régebbi.

```
php log clear 5d
```

6.3.3 Felhasználó kezelés

A parancssorról a felhasználóknak új jelszót tudunk adni, felhasználókat tudunk törölni és új adminisztrátorokat létrehozni.

Adminisztrátor létrehozása

Új adminisztrátor létrehozásához meg kell adnunk az azonosítóját, a nevét és a jelszavát

```
php user create admin Admin password123
```

Felhasználó törlése

Felhasználó törléséhez meg kell adnunk az azonosítóját. Ez a művelet sikertelen ha az adott felhasználó más felhasználók főnöke.

```
php user delete admin
```

Új jelszó adása

Az új jelszó adáshoz meh kell adni a felhasználó azonosítóját és az új jelszót.

```
php user password admin password123
```

6.3.4 Engedélyek

Két féle engedély típus van: a rendszer és a helyi. A rendszer engedélyek az egész rendszerre vonatkoznak. Azaz ha egy felhasználó egy ilyen engedélyt kap akkor az minden telephelyen érvényes, de a rendszer engedélyek között rendszer adminisztrációval kapcsolatosak engedélyek is vannak. A helyi engedélyek telephelyhez vannak kötve. Azaz ha egy felhasználó egy ilyen engedélyt kap akkor csak az adott telephelyen érvényes.

Rendszer engedélyek

- Telephelyek megtekintése
- Telephelyek hozzáadása
- Telephelyek módosítása
- Telephelyek törlése
- Típusok hozzáadása
- Típusok módosítása
- Típusok törlése
- Felhasználók megtekintése
- Felhasználók hozzáadása
- Felhasználók módosítása
- Felhasználók törlése

Helyi engedélyek

- Raktárak kezelése
- Hozzáadó művelet létrehozása
- Eltávolító művelet létrehozása
- Műveletek visszaigazolása

6.3.5 Beállítások

A `config.json` konfigurációs fájl a szerver beállításait tartalmazza.

Adatbázis beállítások

A **database** szekció az alkalmazás adatbázis-kapcsolatához szükséges paramétereket tartalmazza.

host Az adatbázis szerver címe.

user Az adatbázis felhasználó neve.

password Az adatbázis felhasználó jelszava.

database_name Az adatbázis neve, amelyhez csatlakozik az alkalmazás (például: easyinventory).

port Az adatbázis portja, amelyen a kapcsolatot létesíteni kell.

Token lejáratú idő

A **token_expiration** kulcs határozza meg, hogy hány másodpercig érvényes egy generált token. Ez az érték alapértelmezetten 120 másodperc, tehát 2 perc.

Teszt token

A **test_token** kulcs egy fix token-t ad meg, amelyet a rendszer teszteléséhez használhatunk. Ezt normál futtatáskor **null**-ra kell állítani.

Naplázási beállítások

A **logging** szekció az alkalmazás különböző naplázási beállításait tartalmazza. Négy fő típusú naplázás van: **success**, **bad**, **rejected**, és **error**. Mindegyik típusnak van négy különböző beállítása a következő mezők szerint: **request_body**, **response_body**, **debug**, és **error**.

success Sikeres kérések naplózása

bad Hibás kérések naplózása

rejected Elutasított kérések naplózása

error Hiba naplózás

request_body A kérés törzsének naplózása

response_body A válasz törzsének naplózása

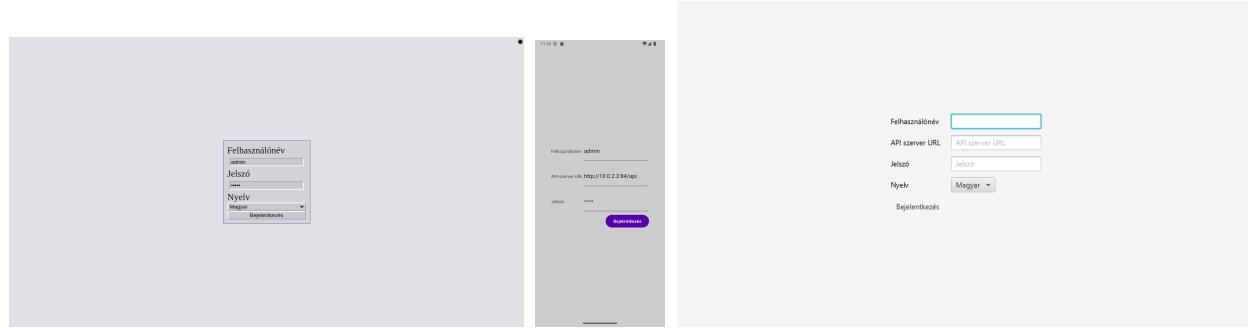
debug A hibakeresési információk naplózása

error A hibák naplózása

6.4 Használat

6.4.1 Bejelentkezés

A program indításakor először a bejelentkező felületre fogad minket, ahol meg kell adni a felhasználónevet, a API szerver url -jét, a jelszót és nyelvet a bejelentkezéshez. Miután kitöltöttük az adatokat, a Bejelentkezés vagy Login gombra kattintva, amennyiben a megadott adatok helyesek, továbbjutunk a kezelőfelületre. A lap bal felső sarkában láthatjuk az alkalmazást használó személy nevét, illetve a mellette található gombal ki is jelentkezhetünk a programból. A lap tetején elhelyezkedő menüelemek segítségével választhatjuk ki, hogy az adatbázis melyik részét szeretnénk megtekinteni vagy módosítani.



6.4.2 Felhasználó kezelés

A Felhasználók menüpont segítségével megtekinthetjük az összes felhasználót, aki hozzáféréssel rendelkezik az alkalmazáshoz. minden felhasználó mellett két gomb található, amelyek segítségével: Módosíthatjuk a felhasználó adatait (név, felhasználónév, jogosultságok, jelszó) Törölhetjük a felhasználót A lap alján található gombokkal: Frissíthetjük az oldalt Új felhasználót regisztrálhatunk.

	Name	Surname	Username	Last login
admin	Alfonso Sloan	Patty Bever	alice_ford203	2023-07-17 07:37
Alfonso Nicholson	Alice Ford	Jenny Dillon	admin	2023-07-17 07:37
Andrea Ward	Alison Franklin	Mario Rivera	alison_franklin1983	2023-07-17 07:37
Antone Juarez	Alva Weiss	Lacy French	alva_weiss2004	2023-07-17 07:37
Antone Juarez	Amelia Valentine	Dona Hendrix	amelia_valentine1989	2023-07-17 07:37
Antone Juarez	Alphonse Melendez	Chris Myers	andrew_finley1974	2023-07-17 07:37
Antone Juarez	Angel Roman	Lizzie Singh	angel_roman2005	2023-07-17 07:37
Antone Juarez	Annabelle Levy	Shirley Yoder	annabelle_levy1999	2023-07-17 07:37
Antone Juarez	Barney Adkins	Lillian Levine	anna1153490288	2023-07-17 07:37
Antone Juarez	Barbara McConnell	Pearlie Frey	barbara_mcconnell1962	2023-07-17 07:37
Antone Juarez	Arnon Silva	Levi Hoover	aron Silva	2023-07-17 07:37
Antone Juarez	Antone Juarez	Davis Hunt	anton_juarez1980	2023-07-17 07:37

Felhasználók	Telephelyek	Termékek	Egyégek	Raktárak	Keresés	Műveletek	Profil
admin							Alfonso Sloan alfonso_sloan1985 Nathan Osborne
Alice Ford							Alice Franklin alice_franklin1983 Nathan Osborne
Alfonso Nicholson							Alphonse Melendez alphonse_melendez1989 Liliana Levine
Alva Weiss							Amelia Valentine amelia_valentine1989 Bernice Odonnell
Amelia Valentine							Andrea Ward andrea_ward1962 Brianna Page
Andrea Ward							Andrew Finley andrew_finley1974 Brianna Page
Angel Roman							Angel Roman angel_roman2005 Lizzie Singh
Annabelle Levy							Anna anna1153490288 admin
Annabelle Levy							Annabelle Levy annabelle_levy1999 Shirley Yoder
Antone Juarez							Antone Juarez anton_juarez1980

Felhasználó hozzáadás

Felhasználó hozzáadásához a gombra kell kattintani. Ezután megjelenik egy bemeneti ablak, ami kitöltése után a gombra nyomhatunk a mentéshez.

Felhasználónév:

Név:

Jelszó:

OK
Cancel

Felhasználónév:

Név:

Jelszó:

OK
Cancel

Felhasználó módosítás

A felhasználó adatait a gombra való kattintással módosíthatjuk. Ezután a megjelent adatlap ki lesz töltve a mentett értékekkel. A módosítás után a gombra kattintással lehet menteni.

Felhasználó engedélyek módosítása

A felhasználó engedélyeit a gombra való kattintással módosíthatjuk. Ilyenkor feljön egy ablak amiben az összes engedély ki van listázva. Itt a gombbal vehetünk el és a gombbal adhatunk engedélyt.

Felhasználó törlés

A felhasználót a gombra való kattintással törölhetjük.

6.4.3 Egység kezelés

Az Egységek menüpont segítségével megtekinthetjük, módosíthatjuk, illetve törölhetjük a nyilvántartásban szereplő, mérésre szolgáló egységeket. A lap alján itt is megtalálhatóak a frissítés és a hozzáadás gombok, amelyekkel frissíthetjük az oldalt, illetve a megfelelő adatok megadásával újabb egységgel bővíthetjük az adatbázist.

Egység hozzáadás

Egy egységet a **+** gombra való kattintással adhatunk hozzá a rendszerhez. Ekkor megjelenik egy bemeneti ablak amit kitöltve **✓** gombra kattintással menthetünk.

Egység módosítás

Egységeket a gombra való kattintással módosíthatunk. Ekkor megjelennek az adatok egy felugró ablak és módosítás után a gombra kattintással menthetünk.

The screenshot shows the unit management application's main interface on the left and an edit dialog on the right. In the edit dialog, the unit 'Hüvelyk' is selected. The 'Azonosító' field contains 'inch1434505200' and the 'Név' field contains 'Hüvelyk'. At the bottom are 'OK' and 'Cancel' buttons.

Egység törlés

Egységeket a gombra való kattintással törölhetünk. Fontos megjegyezni, hogy egy egységet nem törölhetünk ha legalább egy cikk azt használja.

The screenshot shows the unit management application's main interface on the left and a confirmation dialog on the right. The dialog asks 'Biztosan ki szeretné törölni?' (Are you sure you want to delete?) with 'OK' and 'Cancel' buttons. A question mark icon is also present.

6.4.4 Árucikkek kezelése

Az árucikkek menüpont segítségével megtekinthetjük, módosíthatjuk, illetve törölhetjük a nyilvántartásban szereplő, árucikkeket. A lap alján itt is megtalálhatóak a frissítés és a hozzáadás gombok, amelyekkel frissíthetjük az oldalt, illetve a megfelelő adatok megadásával újabb cikkekkel bővíthetjük a rendszert.

The screenshot shows two windows side-by-side. The left window is titled 'Termékek' (Products) and lists various steel wire products like 'Acél 10mm', 'Acélkábel 10mm', etc., each with a 'Delete' icon. The right window is titled 'Felhasználók' (Users) and shows a list of users with their roles and profile icons.

Árucikk hozzáadás

Egy árucikkek a **+** gombbal adhatunk hozzá a rendszerhez. Ekkor megjelenik egy bemeneti ablak amit kitöltve **✓** gombra kattintással menthetünk.

The screenshot shows the product addition dialog box. It has fields for 'Azonosító' (Identifier), 'Név' (Name), and 'Mértékegység' (Unit). Below these is a note: 'Nincs kiválasztott egység' (No unit selected). At the bottom are 'OK' and 'Cancel' buttons.

Árucikk módosítás

Árucikkeket a ✎ gombra való kattintással módosíthatunk. Ekkor megjelennek az adatok egy felugró ablak és módosítás után a ✓ gombra kattintással menthetünk.

The screenshot shows a list of products on the left and a detailed edit dialog on the right. The products listed include: Acél (steel), Acélkábel 10mm (steel wire_10), Acélkábel 12mm (steel wire_12), Acélkábel 13mm (steel wire_13), Acélkábel 16mm (steel wire_16), Acélkábel 20mm (steel wire_20), Acélkábel 25mm (steel wire_25), Acélkábel 35mm (steel wire_35), Acélkábel 3mm (steel wire_3), Acélkábel 40mm (steel wire_40), Acélkábel 4mm (steel wire_4), Acélkábel 5mm (steel wire_5), Acélkábel 8mm (steel wire_8), Ácskapocs (nail), Agyag téglá (cement tile), and Alma (apple). The edit dialog on the right shows fields for Azonosító (Identifier) set to computer1434505200, Név (Name) set to Számítógép (Computer), and Mértékegység (Unit) set to Darab (Piece). Buttons for OK and Cancel are at the bottom.

Árucikk törlés

Árucikkeket a 🗑 gombra való kattintással törölhetünk. Fontos megjegyezni, hogy egy árucikket nem törölhetünk ha van valahol ilyen tárolva a rendszerben.

The screenshot shows a list of products on the left and a confirmation dialog on the right. The products listed are identical to the previous screenshot. The confirmation dialog asks "Biztosan ki szeretné törölni?" (Are you sure you want to delete?). Buttons for OK and Cancel are at the bottom. A help icon is also present.

6.4.5 Telephely kezelés

Az telephelyek menüpont segítségével megtekinthetjük, módosíthatjuk, illetve törölhetjük a nyilvántartásban szereplő, telephelyeket. A lap alján itt is megtalálhatóak a frissítés és a hozzáadás gombok, amelyekkel frissíthetjük az oldalt, illetve a megfelelő adatok megadásával újabb telephellyel bővíthetjük a rendszert.

Telephely hozzáadás

Egy telephelyet a **+** gombbal adhatunk hozzá a rendszerhez. Ekkor megjelenik egy bemeneti ablak amit kitöltve **✓** gombra kattintással menthetünk.

Telephely módosítás

Egy telephelyet a gombra való kattintással módosíthatunk. Ekkor megjelennek az adatok egy felugró ablak és módosítás után a gombra kattintással menthetünk.

The screenshot shows a list of locations on the left and a detailed edit dialog on the right. The edit dialog contains fields for 'Azonosító' (Identifier) set to 'WH_sátoraljaújhely14345', 'Név' (Name) set to 'Sátoraljaújhelyi telephely', and 'Cím' (Address) set to 'nekeresd utca 1.'. There are 'OK' and 'Cancel' buttons at the bottom right of the dialog.

Telephely törlés

Telephelyet a gombra való kattintással törölhetünk. Fontos megjegyezni, hogy egy telephelyet nem törölhetünk ha van raktár benne. Azaz először a raktárakat kell törölni.

The screenshot shows a list of locations on the left and a confirmation dialog on the right. The dialog asks 'Biztosan ki szeretné törölni?' (Are you sure you want to delete?). It has 'OK' and 'Cancel' buttons at the bottom right.

6.4.6 Raktár kezelés

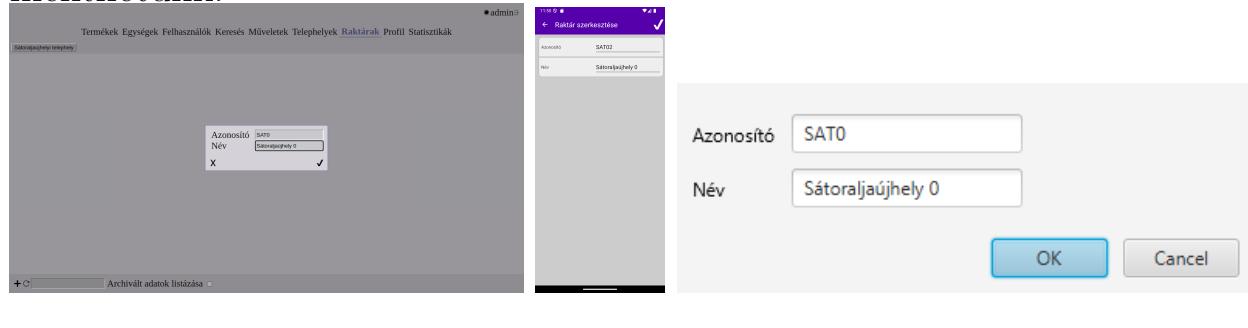
A Raktárok menüpontra kattintva a program megmutatja számunkra a raktárokat telephelyenként szétválogatva. A menüsor alatt található lenyílő listával tudjuk kiválasztani, hogy melyik telephelyhez tartozó raktárokat szeretnénk megtekinteni, majd a raktárok neve mellett található gombokkal módosíthatjuk és törölhetjük azokat. A lap alján elhelyezkedő, hozzáadás gombbal pedig új raktárt adhatunk hozzá a kiválasztott telephelyhez.

Raktár hozzáadás

Egy telephelyet a **+** gombbal adhatunk hozzá a rendszerhez. Ekkor megjelenik egy bemeneti ablak amit kitöltve **✓** gombra kattintással menthetünk.

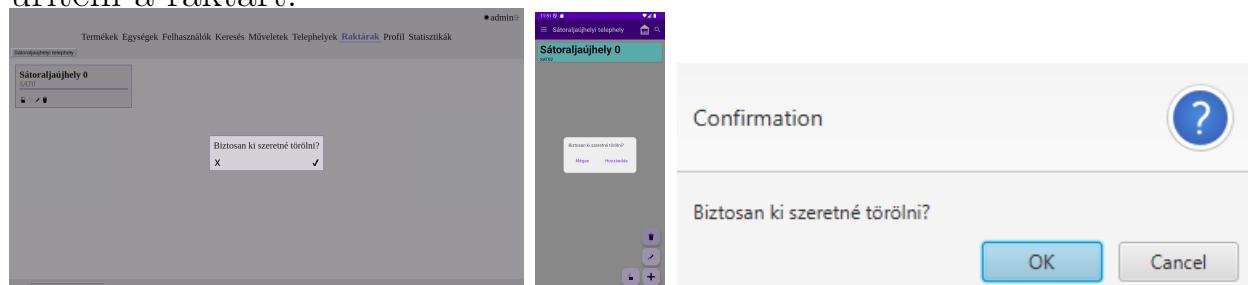
Raktár módosítás

Egy raktárt a gombra való kattintással módosíthatunk. Ekkor megjelennek az adatok egy felugró ablak és módosítás után a gombra kattintással menthetünk.



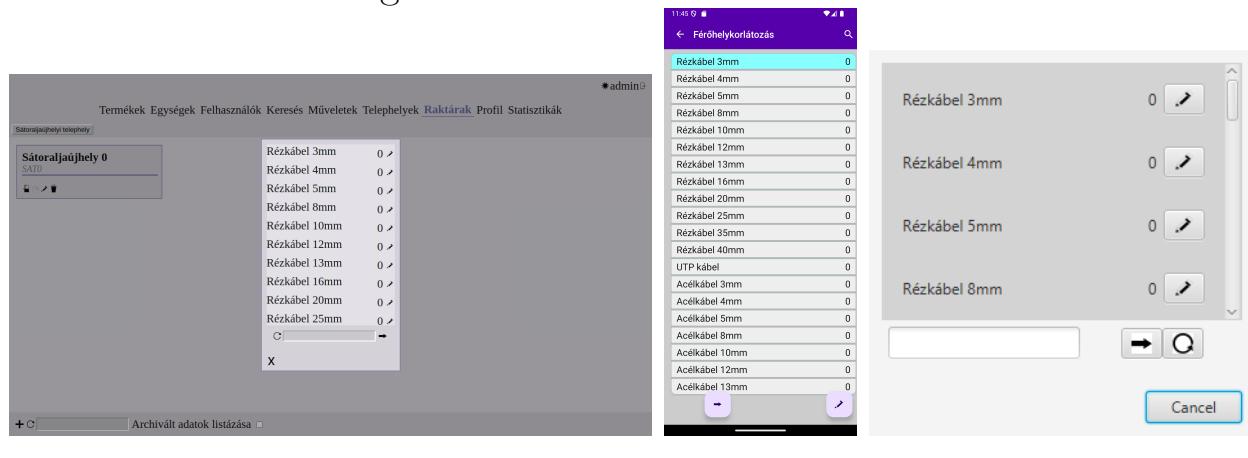
Raktár törlés

Raktárakat a gombra való kattintással törölhetünk. Fontos megjegyezni, hogy egy raktárt nem törölhetünk ha van benne áru. Azaz először ki kell üríteni a raktárt.



Raktár limitek

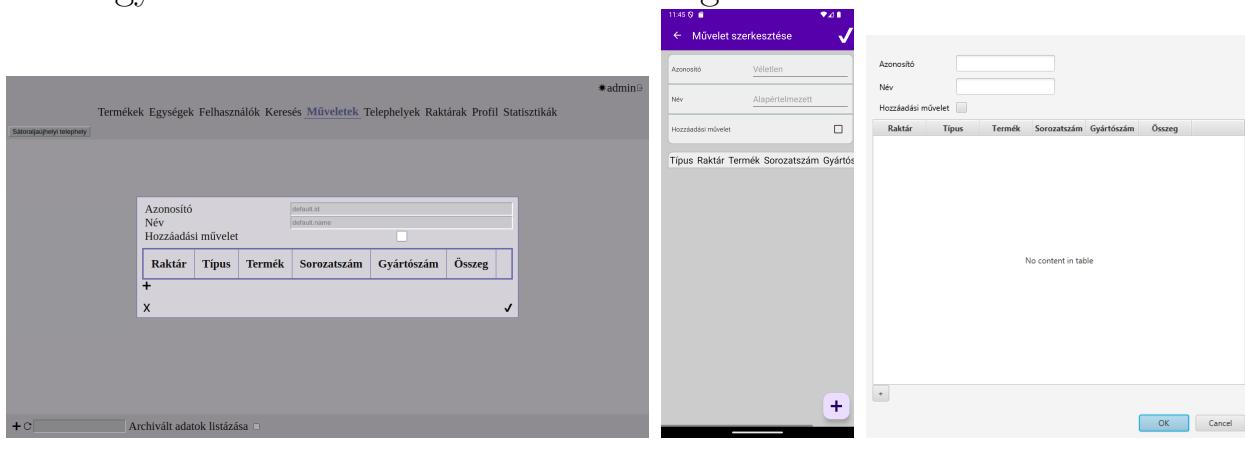
Raktár limiteket a gombra való kattintással szerkeszthetünk.



6.4.7 Műveletek

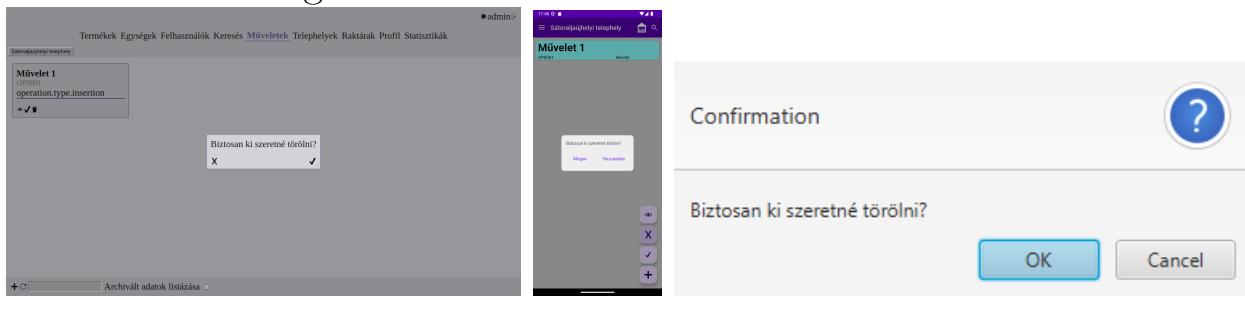
Művelet hozzáadás

Egy műveletet a **+** gombbal adhatunk hozzá a rendszerhez. Ekkor megjelenik egy bemeneti ablak amit kitöltve **✓** gombra kattintással menthetünk.



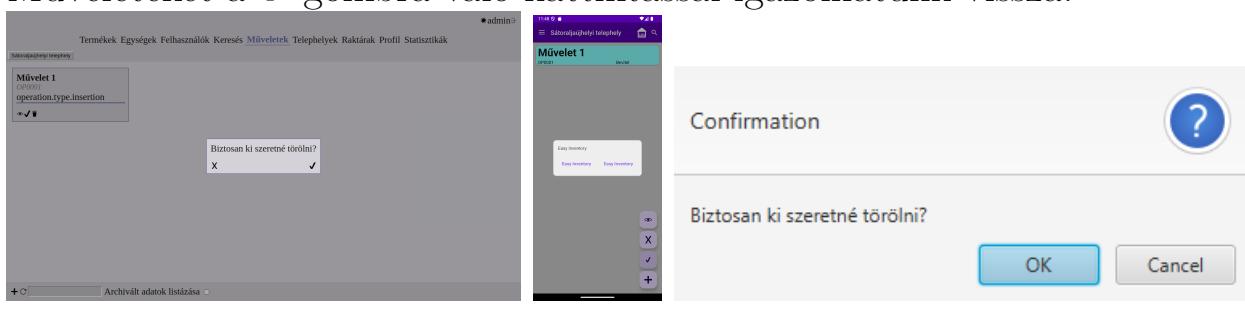
Művelet törlés

Műveleteket a **ⓧ** gombra való kattintással törölhetünk.



Művelet visszaigazolás

Műveleteket a **✓** gombra való kattintással igazolhatunk vissza.



6.4.8 Keresés

A Keresés használatával egyszerűen rá tudunk keresni bármilyen termékre, a program pedig visszaadja, hogy melyik telephelynek melyik raktárában található meg az adott termék, illetve további információkat találunk, mint például a rendelkezésre álló mennyiség vagy a termék sorozatszáma.

The screenshot shows two windows side-by-side. The left window is titled 'Keresés' and displays a table of search results:

Típus	Mennyisége	Elérhető
Acélkábel 16mm	789 Méter	706 Méter
Cement	888 Kilogramm	802 Kilogramm
Ferdehátú autó	36 Darab	36 Darab
huzalszeg	1417 Doboz 25x25x25	1323 Doboz 25x25x25
Rézkábel 12mm	772 Méter	721 Méter
Zöld körállakú szőnyeg	1261 Négyzetméter	1222 Négyzetméter
Bab	990 Konzervdoboz	873 Konzervdoboz
Barna csíkos szőnyeg	1092 Négyzetméter	1040 Négyzetméter

The right window shows a detailed view of the search results for 'Keresés' with the following table:

Típus	Mennyisége	Elérhető
Acélkábel 1...	619	561
Plátisz teherautó	55	55
Rézkábel 3mm	976	976
Benzin (95)	1089	1089
Fold	757	757
Kukorica	1446	1446
Piros kockás szőnyeg	1644	1644
Rézkábel 13mm	774	774
Benzin (E85)	1393	1393
Teherszám	32	32
Szelekció autó	44	44
Aszfaltkábel 4mm	942	942
Piros hosszú szálú szőnyeg	889	889
Acélkábel 40mm	1109	1109
Vértelebor	1175	1175
Naranccs	1180	1180
Fekete hosszú szálú szőnyeg	1092	1092
Zöld polifoam szőnyeg	1132	1132
Acélkábel 5mm	814	814
Kötél (gr... "szálén")	1184	1184
LNG	753	753

6.5 Használati példák

6.5.1 Fatelep

Tegyük fel, hogy egy favágó vállalatunk több fatelephellyel rendelkezik. Mivel a fakitermelés és annak értékesítése folyamatos folyamat, így a telephelyeken található fa mennyisége is rendszeresen változik. A telephelyek területe véges, ebből kifolyólag elengedhetetlen egy olyan nyilvántartó rendszer alkalmazása, amely pontosan követi, hogy egyes telepeken mennyi fa tárolására van még lehetőség. Emellett mivel a fa folyamatos értékesítése zajlik, fontos, hogy nagyobb megrendelések esetén ne fogadjunk el olyan rendelést, amelyhez nincs elegendő fa készletünk.

Amennyiben a fatelep feldolgozással is foglalkozik, és különböző formájú (pl. rönk, hasáb, deszka, gerenda) és méretű faanyagokkal dolgozik, azok megfelelő nyilvántartása és kezelésük egyszerűsítése érdekében alkalmazásunk ideális megoldást kínál. A rendszer lehetővé teszi a különféle faelemek pontos nyomon követését, biztosítva ezzel a hatékony és átlátható működést.

6.5.2 Bevásárló központ

Ebben a példában egy bevásárlóközpont üzemeltetésével foglalkozunk, amely több üzlettel és különböző területeken található szolgáltatásokkal rendelkezik. A központ forgalma és a rendelkezésre álló árukészletek folyamatosan változnak, így szükség van egy olyan nyilvántartó rendszerre, amely pontosan követi a termékek és készletek állapotát, valamint a raktárkapacitásokat is. A vásárlók folyamatosan fogyasztják a boltok kínálatát, így fontos tudni róla, ha elfogy, vagy fogyóban valamelyik cikk és utánpótlást kell küldeni.

A nyilvántartás kezelő rendszerünkben munkakörnek megfelelő jogosultságokkal lehet ellátni az egyes felhasználókat, ezzel elkerülve, hogy valaki olyat tegyen, amire nincs felhatalmazása. Az alkalmazásunk ideális választás a központ napi működésének zökkenőmentes irányításához, segítve a készletek, üzletek és szolgáltatások hatékony kezelését és nyilvántartását.

Továbbfejlesztési lehetőségek

7.1 Adminisztráció

7.1.1 Integráció monitorozó rendszerekkel

Integráció monitorozó rendszerekkel mint például Grafana. Ezen keresztül lehetne nézni a logokat.

7.1.2 Központi beléptető rendszer

A központi beléptető rendszer megkönnyítené a felhasználókezelést és segítene beintegrálni a rendszert nagyobb vállalatok rendszerébe.

7.1.3 Webhook-ok

Webhook-okat lehetne létrehozni, hogy különböző folyamatokat elindítson ha történek egy bizonyos esemény.

7.2 Statisztika

7.2.1 Kördiagram telephely/raktár tartalmáról

Egy kördiagram ami mutatná a raktár tartalmát és miből milyen arányban van. Lehetne darabszám vagy elfoglalt hely alapján is elkészítve a diagram

7.2.2 Különböző raktárak/telephelyek összehasonlítása

Lenne egy táblázat ami összehasonlítja a raktárak vagy telephelyek tartalmát árucikkek szerint Könnyebb lenne átlátni és rendszerezni a raktárat

7.3 Használat

7.3.1 Térkép a raktárról és benne a tárgyakról

Amikor egy konkrét tárgyra rákeresünk akkor egy térkép megjelenne és lehetne látni hogy a raktáron belül hol találhatóak a tárgyak

7.3.2 Rendszeres művelet order

Létre lehetne hozni rendszeresített műveletet ami a megadott időközönként létrehoz egy műveletet. A hozzá tartozó tárgyak polc számát automatikusan kiszámolná.

7.3.3 Mértékegység átváltás

Mértékegységeket át lehessen váltani, hogy a raktár telítettségét jobban meg lehessen mondani, és könnyebb legyen kezelní a raktár tárolókapacitását.

7.3.4 Raktártípusok

Raktártípusokat lehessen létrehozni és beállítani. Ennek köszönhetően amikor új raktárat hozunk létre akkor az összes korlátozás és egyéb dolog automatikusan be lenne állítva.

7.3.5 Riasztás beállítás

Riasztást lehetne beállítani ha hamarosan elfogy valami.

7.3.6 QR kód olvasó

QR kód olvasó a telefonos alkalmazáshoz: A mobilalkalmazásban QR kódok olvasásának beépítése, ami egyszerűsítheti a felhasználók számára az információk gyors elérését, illetve felvitelét.

Felhasznált irodalom

- <https://developer.mozilla.org>
- <https://www.php.net/docs.php>
- <https://playwright.dev/docs/intro>
- <https://developer.android.com/training/testing/espresso>

Mellékletek

9.1 Felhasznált könyvtárak

- Playwright
- <https://github.com/stleary/JSON-java>

9.2 Online elérhetőség

| TODO