Міністерство освіти та науки України
Харківський національний університет радіоелектроніки
Кафедра програмної інженерії

Лабораторна робота №2

З дисципліни: «Архітектура програмного забезпечення»

на тему: «Програмна система для автоматизації процесів прибирання приміщень»

Виконав
ст. гр. ПЗПІ-18-2
Кузнецов Роман Олександрович

Перевірив
ст. викл. каф. ПІ
Сокорчук І. П.

Харків 2021

Мета: розробити серверну частину програмної системи, описати прийняті інженерні рішення, будову серверних компонентів, загальну структуру системи та структуру бази даних.

Хід роботи:

Каскад серверу реалізований повністю за допомогою фреймворку Spring Boot. Для написання класів-контролерів, створення кінцевих точок, реалізації REST API сервісів для взаємодії з клієнтами використовувався Spring WebFlux – фреймворк для реалізації web-додатків на мовах, що використовують JVM. Доступ до даних та реалізація ORM написана за допомогою Spring Data JPA та Hibernate. Для захисту системи, реалізації рівнів доступу, авторизації використано фреймворк Spring Security. Обрана система управління базами даних для проекту – MySQL.

Перед початком створення серверної частини було спроектовано схему бази даних та створені зв’язки між сутностями в ній. ER-діаграма для програмної системи зображена на рисунку 1.

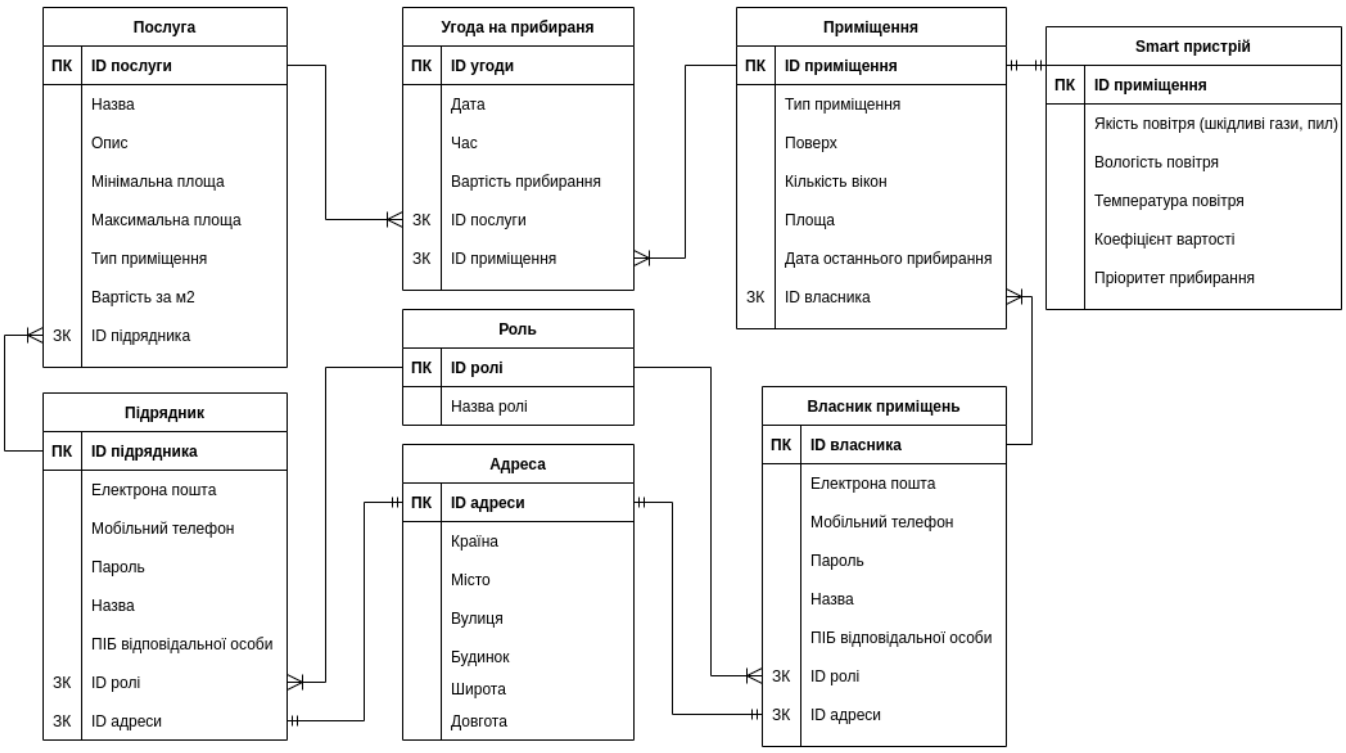


Рисунок 1 – ER-модель даних

Далі було розпочато реалізацію класів, які відображають сутності БД у коді програмної системи з використанням Java Persistence Api, стандарту проектування ORM для Spring Data.

Отримано 9 класів сутностей за стандартом JPA. Приклад програмної реалізації класу PlacementOwner, який відображає сутність «Власник приміщень»:

```
@Entity
@Data
public class PlacementOwner {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String phoneNumber;

    private String name;

    private String email;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "address_id",
        referencedColumnName = "address_id")
    private Address address;

    @OneToMany(mappedBy = "placementOwner", fetch = FetchType.EAGER)
    @OnDelete(action = OnDeleteAction.CASCADE)
    private Set<Placement> placements;

}
```

Наступним кроком було створення репозиторіїв, які надають API для доступу та маніпулювання даними програмної системи, що зберігаються у СКБД MySQL. Всі репозиторії розширюють функціонал інтерфейсу з параметрами CrudRepository фреймворку Spring Data JPA, надають базовий доступ до даних, Spring-компонент «Repository» для впровадження у сервісний шар та можливість написання власних методів вибірки.

Приклад програмної реалізації інтерфейсу AddressRepository:

```
@Repository
public interface PlacementOwnerRepository extends
CrudRepository<PlacementOwner, Long> {
    Optional<PlacementOwner> findByPhoneNumber(String phoneNumber);
    Optional<PlacementOwner> findByEmail(String email);
    List<PlacementOwner> findAll();
}
```

Сукупність цих інтерфейсів являє собою перший шар серверного рівню системи – шар доступу до даних. В ньому зосереджена лише логіка роботи з базою даних.

Далі було реалізовано сервісний шар – шар, у якому реалізується бізнес-логіка програмної системи. Таке розподілення програмних компонентів робить систему більш гнучкою для доповнення та масштабування. Приклад програмної реалізації логіки регулювання вартості складених угод на прибирання приміщення наведено у додатку А. Приклад програмної реалізації бізнес-логіки автоматичного розрахунку вартості праці підрядного працівника наведено у додатку Б.

Наступний шар програми - це шар контролерів. Головною задачею контролерів є прийом та обробка HTTP запитів, виклик методів нижчих шарів системи. Було реалізовано мережеве REST API для взаємодії з веб-клієнтом, мобільним застосунком та IoT пристроєм з серверною частиною системи.

Було створено діаграму компонентів та діаграму розгортання, які наведені у додатках В та Г відповідно.

На діаграмі компонентів вказані усі компоненти трьох шарів системи. Ці ж самі компоненти системи зображені в серверному вузлу на діаграмі розгортання, яка демонструє загальну архітектуру системи.

Після написання логіки було запроваджено захист системи за допомогою фреймворку Spring Security. Було реалізовано логіку маніпулювання JWT токеном. Він включає у себе пошту користувача та його права. На основі цих даних відбувається як аутентифікація, так і авторизація з подальшим відкриттям доступу до захищених URL системи.

У структурі проекту також наявні два допоміжні пакети класів. Перший пакет зберігає в собі DTO (Data Transfer Object) – об’єкти передачі даних, які приймаються у тілі запиту від клієнтів або надсилаються до них у тілі відповіді сервера.

У другому пакеті зосереджена логіка перевірки коректності надісланих від клієнтів даних. Ця перевірка могла б бути реалізована лише клієнтами, але вона

реалізована і на стороні сервера для збільшення безпеки. Таким чином стають неможливими ситуації, коли до бази даних заносяться неправильні, небезпечні дані.

Після реалізації серверної частини проекту була створена загальна діаграма варіантів використання, наведена у додатку Д. Специфікація REST у форматі API Blueprint Format 1A наведена у додатку Е.

Посилання на архів з програмним кодом та файлом контрольної суми: https://drive.google.com/drive/folders/1TpkrPkFM1t04XNKByazjfc-D0a9uWiut?usp=sharing

Контрольна сума до архіву: 69642801caacae9c70efeca63070276b

Висновок: у ході лабораторної роботи було створено серверну частину програмної системи. Було описано та обґрунтовано прийняті інженерні рішення, будову серверних компонентів, загальну структуру системи та структуру бази даних. Також у ході виконання було створено наступні діаграми - UML діаграму розгортання, UML діаграму прецедентів, ER-модель даних та UML діаграму компонентів.

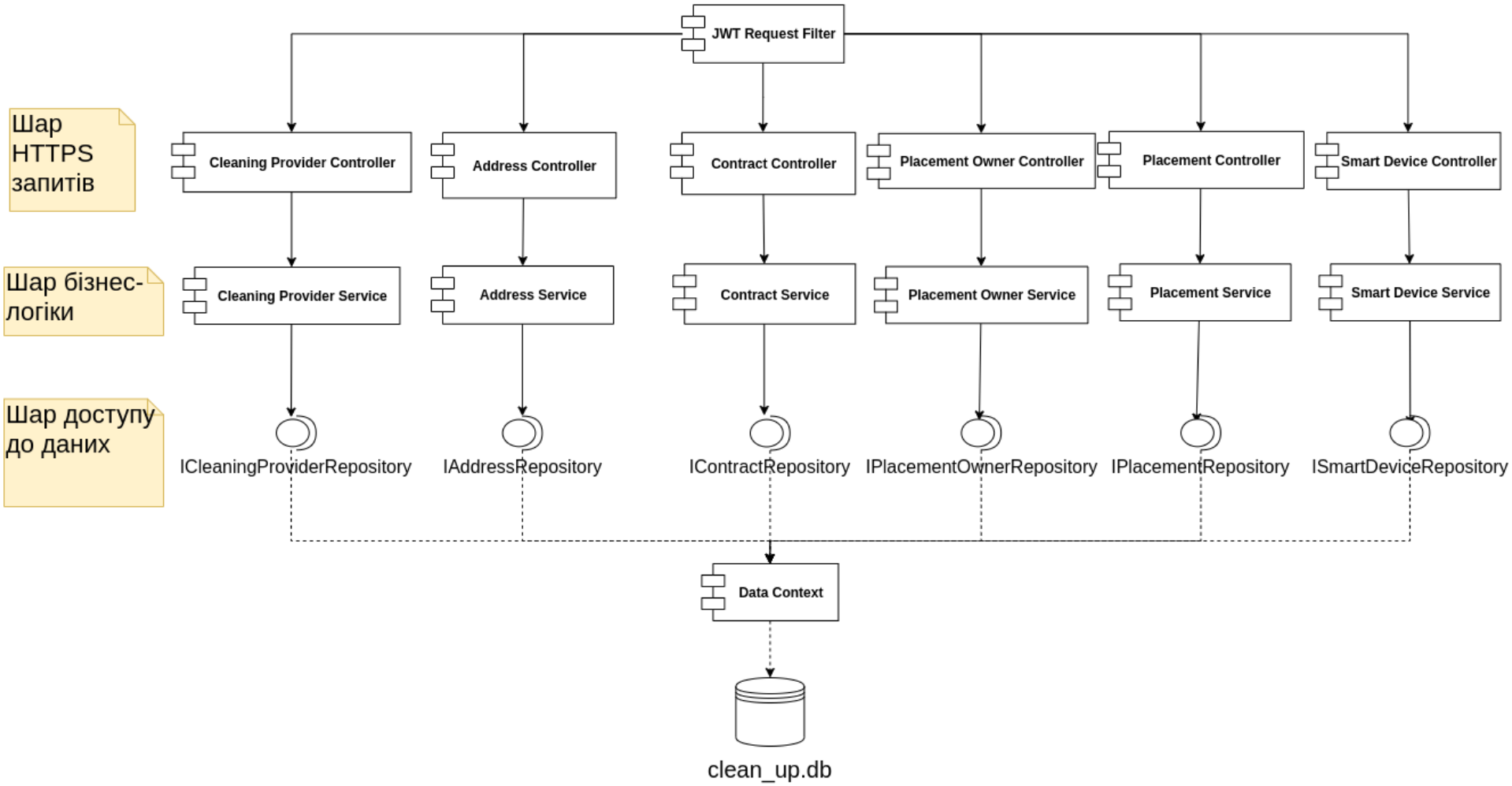
ДОДАТОК А. Програмна реалізація регулювання вартості складених угод

```
@Override
public PlacementDto updateSmartDevice(SmartDeviceDto smartDeviceDto) {
    Optional<Placement> placementDto = placementRepository
            .findById(smartDeviceDto.getId());

    if (placementDto.isPresent()) {
        Placement placement = placementDto.get();

        SmartDevice smartDevice = placement.getSmartDevice();
        double previousAdjustmentFactor =
                smartDevice.getAdjustmentFactor();
        double adjustmentFactor =
                round(smartDeviceDto.getAdjustmentFactor());

        Date currentDate = new Date();
        placement.getAgreements().stream()
                .filter(
                        contract -> contract.getDate().after(currentDate)
                )
                .forEach(contract -> {
                    double price = contract.getPrice();

                    if (previousAdjustmentFactor != 0) {
                        price /= previousAdjustmentFactor;
                    }

                    if (adjustmentFactor != 0) {
                        price *= adjustmentFactor;
                    }

                    contract.setPrice(round(price));
                });

        smartDevice
                .setAirQuality(smartDeviceDto.getAirQuality())
                .setTemperature(smartDeviceDto.getTemperature())
                .setHumidity(smartDeviceDto.getHumidity())
                .setAdjustmentFactor(adjustmentFactor)
                .setDirtinessFactor(smartDeviceDto.getDirtinessFactor())
                .setPriority(smartDeviceDto.getPriority());
        placement.setSmartDevice(smartDevice);

        return PlacementMapper
                .toPlacementDto(placementRepository.save(placement));
    }

    return null;
}
```

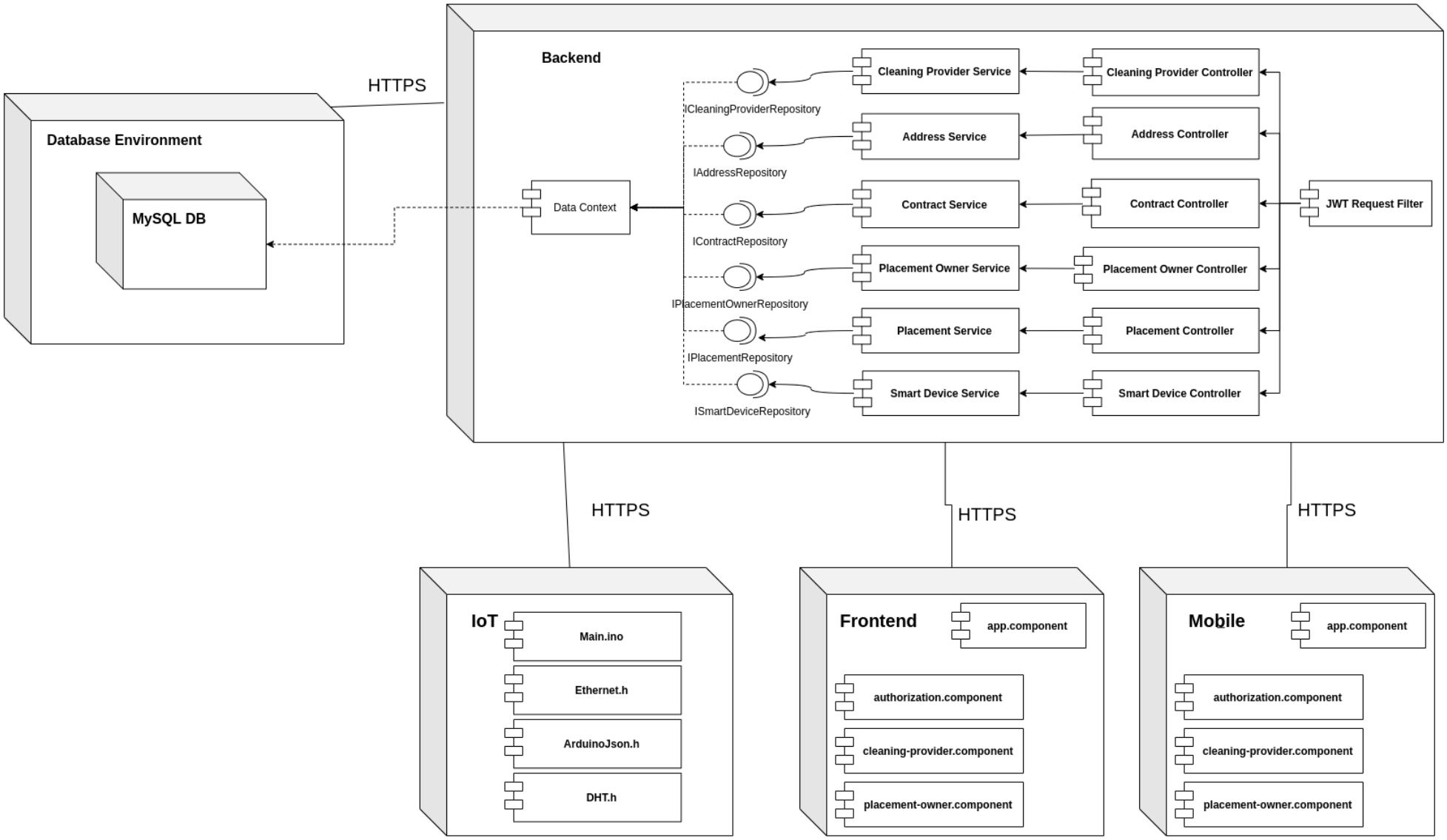
Додаток Б. Програмна реалізація розрахунку вартості праці підрядника

```
@Override
public AgreementDto create(PriceDto priceDto) {
    Agreemnt agreement = new Agreement();
    agreement.setDate(priceDto.getDate());

    Optional<Placement> placementById = placementRepository
            .findById(priceDto.getPlacementId());
    Optional<ProviderService> providerServiceById =
            providerServiceRepository.findById(
                    priceDto.getProviderServiceId()
            );

    if (placementById.isPresent() && providerServiceById.isPresent()) {
        Placement placement = placementById.get();
        ProviderService providerService = providerServiceById.get();

        agreement.setPlacement(placement);
        agreement.setProviderService(providerService);

        double price =
                providerService.getPricePerMeter() * placement.getArea();
        Double priceFactor = null;

        SmartDevice smartDevice = placement.getSmartDevice();

        if (isPresent(smartDevice))
            priceFactor = smartDevice.getAdjustmentFactor();

        if (isPresent(priceFactor)) {
            if (priceFactor != 0) {
                price = price * priceFactor;
            }
        }
        agreenent.setPrice(price);

        return AgreemntMapper.toAgreemntDto(
                agreementRepository.save(agreement)
        );
    }

    return null;
}
```

Додаток В. Діаграма компонентів системи



Додаток Г. Діаграма розгортання системи



Додаток Д. Діаграма варіантів використання системи

ДОДАТОК Е. Специфікація REST у форматі API Blueprint Format 1A

```
FORMAT: 1A
HOST: http://localhost:8080/

# Backend system API
Documentation for REST API endpoints

## Authentication
This API uses Custom Header for its authentication.

The parameters that are needed to be sent for this type of authentication are as
follows:
+ `Authentication`

# Group Admin

Admin Controller

## Admin Backup [/admin/backup]

### getBackupData [GET]
Performs data backup and returns mysql dump file

+ Response 200 (text/plain)

        OK

    + Attributes (object)


+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found



# Group Provider

Provider Controller

## Cleaning Providers [/cleaning-providers]

### getAllCleaningProviders [GET]
Returns a list of all Providers

+ Response 200 (text/plain)

        OK

    + Attributes (object)
```

+ Response 401

Unauthorized

+ Response 403

Forbidden

+ Response 404

Not Found

addCleaningProvider [POST]
Adds new Provider

+ Request (application/json)

+ Attributes (CleaningProviderDto)

+ Response 200 (text/plain)

OK

+ Attributes (object)

+ Response 201

Created

+ Response 401

Unauthorized

+ Response 403

Forbidden

+ Response 404

Not Found

updateCleaningProvider [PUT]
Updates the Provider

+ Request (application/json)

+ Attributes (CleaningProviderDto)

+ Response 200 (text/plain)

OK

```
    + Attributes (object)

+ Response 201

        Created

+ Response 401

        Unauthorized

+ Response 403

        Forbidden

+ Response 404

        Not Found


## Cleaning Providers Services By Id [/cleaning-providers/services/{id}]

+ Parameters
    + id (number, required)

        id

### getProviderServiceById [GET]
Finds provider service by id

+ Response 200 (text/plain)

        OK

    + Attributes (object)

+ Response 401

        Unauthorized

+ Response 403

        Forbidden

+ Response 404

        Not Found

### deleteProviderService [DELETE]
Deletes provider service by ID

+ Response 200

        OK
```

```
+ Response 204

        No Content


+ Response 401

        Unauthorized


+ Response 403

        Forbidden


## Cleaning Providers By Email [/cleaning-providers/{email}]

+ Parameters
    + email (string, required)

        email


### getCleaningProviderByEmail [GET]
Finds Provider by email

+ Response 200 (text/plain)

        OK

    + Attributes (object)


+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found


### deleteCleaningProvider [DELETE]
Deletes Provider by email

+ Response 200

        OK


+ Response 204

        No Content
```

```
+ Response 401

        Unauthorized


+ Response 403

        Forbidden


## Cleaning Providers Services By Email [/cleaning-providers/{email}/services]

+ Parameters
    + email (string, required)

        email


### getAllProviderServices [GET]
Returns all Provider services (offers)

+ Response 200 (text/plain)

        OK

    + Attributes (object)


+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found


### addProviderService [POST]
Adds new service for Provider

+ Request (application/json)

    + Attributes (ProviderServiceDto)

+ Response 200 (text/plain)

        OK

    + Attributes (object)

+ Response 201

        Created
```

```
+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found


### updateProviderService [PUT]
Updates service of Provider (service ID must be present!)

+ Request (application/json)

    + Attributes (ProviderServiceDto)

+ Response 200 (text/plain)

        OK

    + Attributes (object)

+ Response 201

        Created


+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found


# Group Agreement

Agreement Controller

## Agreements [/Agreements]

### addAgreement [POST]
Creates new Agreement

+ Request (application/json)

    + Attributes (AgreementRequestDto)
```

```
+ Response 200 (text/plain)

        OK

    + Attributes (object)

+ Response 201

        Created


+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found


### updateAgreement [PUT]
Updates Agreement (Agreement ID must be present, updates only date)

+ Request (application/json)

    + Attributes (AgreementRequestDto)

+ Response 200 (text/plain)

        OK

    + Attributes (object)


+ Response 201

        Created


+ Response 401

        Unauthorized

+ Response 403

        Forbidden


+ Response 404

        Not Found
```

```
## Agreements Cleaning Provider By Email [/Agreements/cleaning-provider/{email}]

+ Parameters
    + email (string, required)

        email


### getAllAgreementsByCleaningProvider [GET]
Finds all Agreements for Provider

+ Response 200 (text/plain)

        OK

    + Attributes (object)

+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found


## Agreements Placement Owner By Email [/Agreements/placement-owner/{email}]

+ Parameters
    + email (string, required)

        email


### getAllAgreementsByPlacementOwner [GET]
Finds all Agreements for placement owner

+ Response 200 (text/plain)

        OK

    + Attributes (object)


+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found
```

```
## Agreements By Id [/Agreements/{id}]

+ Parameters
    + id (number, required)

        id


### getAgreementById [GET]
Finds Agreement by id

+ Response 200 (text/plain)

        OK

    + Attributes (object)


+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found


### deleteAgreement [DELETE]
Deletes Agreement by ID

+ Response 200

        OK


+ Response 204

        No Content


+ Response 401

        Unauthorized


+ Response 403

        Forbidden



# Group Placement Owner

Placement Owner Controller

## Placement Owners [/placement-owners]
```

```
### getAllPlacementOwners [GET]
Returns a list of all placement owners

+ Response 200 (text/plain)

        OK

    + Attributes (object)

+ Response 401

        Unauthorized


+ Response 403

        Forbidden

+ Response 404

        Not Found


### addPlacementOwner [POST]
Adds new placement owner

+ Request (application/json)

    + Attributes (PlacementOwnerDto)


+ Response 200 (text/plain)

        OK

    + Attributes (object)


+ Response 201

        Created


+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found


### updatePlacementOwner [PUT]
Updates the placement owner

+ Request (application/json)
```

```
    + Attributes (PlacementOwnerDto)

+ Response 200 (text/plain)

        OK

    + Attributes (object)

+ Response 201

        Created

+ Response 401

        Unauthorized

+ Response 403

        Forbidden

+ Response 404

        Not Found

## Placement Owners Placements By Id [/placement-owners/placements/{id}]

+ Parameters
    + id (number, required)

        id

### getPlacementById [GET]
Finds placement by id

+ Response 200 (text/plain)

        OK

    + Attributes (object)

+ Response 401

        Unauthorized

+ Response 403

        Forbidden

+ Response 404

        Not Found
```

```
### deletePlacement [DELETE]
Deletes placement by ID

+ Response 200

        OK

+ Response 204

        No Content


+ Response 401

        Unauthorized


+ Response 403

        Forbidden


## Placement Owners By Email [/placement-owners/{email}]

+ Parameters
    + email (string, required)

        email

### getPlacementOwnerByEmail [GET]
Finds placement owner by email

+ Response 200 (text/plain)

        OK

    + Attributes (object)


+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found


### deletePlacementOwner [DELETE]
Deletes placement owner by email

+ Response 200

        OK
```

+ Response 204

No Content

+ Response 401

Unauthorized

+ Response 403

Forbidden

Placement Owners Placements By Email [/placement-owners/{email}/placements]

+ Parameters
+ email (string, required)

email

getAllPlacements [GET]
Returns all placements

+ Response 200 (text/plain)

OK

+ Attributes (object)

+ Response 401

Unauthorized

+ Response 403

Forbidden

+ Response 404

Not Found

addPlacement [POST]
Adds new placement for owner

+ Request (application/json)

+ Attributes (PlacementDto)

+ Response 200 (text/plain)

OK

+ Attributes (object)

+ Response 201

Created

+ Response 401

Unauthorized

+ Response 403

Forbidden

+ Response 404

Not Found

updatePlacement [PUT]
Updates placement owner (placement id must be present)

+ Request (application/json)

+ Attributes (PlacementDto)

+ Response 200 (text/plain)

OK

+ Attributes (object)

+ Response 201

Created

+ Response 401

Unauthorized

+ Response 403

Forbidden

+ Response 404

Not Found

 # Group Role

Role Controller

Roles [/roles]

getAllRoles [GET]
Returns a list of all roles

```
+ Response 200 (text/plain)

        OK

    + Attributes (object)

+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found


# Group IoT

IoT Controller

## Device [/device]

### updateSmartDevice [POST]
Update IoT characteristics, endpoint for Arduino

+ Request (application/json)

    + Attributes (SmartDeviceDto)


+ Response 200 (text/plain)

        OK

    + Attributes (object)


+ Response 201

        Created


+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found
```

```
# Group Loginentication

## Login Login [/Login/login]

### loginUser [POST]
Performs user login to the system

+ Request (application/json)

    + Attributes (LoginDto)

+ Response 200 (text/plain)

        OK

    + Attributes (object)


+ Response 201

        Created


+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found


## Login Register Cleaning Provider [/Login/register/cleaning-provider]

### registerCleaningProvider [POST]
Registers a new Provider

+ Request (application/json)

    + Attributes (CleaningProviderDto)


+ Response 200 (text/plain)

        OK

    + Attributes (object)


+ Response 201

        Created
```

```
+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found


## Login Register Placement Owner [/Login/register/placement-owner]

### registerPlacementOwner [POST]
Registers a new placement owner

+ Request (application/json)

    + Attributes (PlacementOwnerDto)


+ Response 200 (text/plain)

        OK

    + Attributes (object)


+ Response 201

        Created


+ Response 401

        Unauthorized


+ Response 403

        Forbidden


+ Response 404

        Not Found


# Data Structures

## AddressDto (object)


### Properties
+ `city` (string, optional)
+ `country` (string, optional)
+ `houseNumber` (string, optional)
+ `latitude` (string, optional)
+ `longitude` (string, optional)
```

```
+ `street` (string, optional)


## CleaningProviderDto (object)


### Properties
+ `address` (AddressDto, optional)
+ `creationDate` (string, optional)
+ `email` (string, optional)
+ `id` (number, optional)
+ `name` (string, optional)
+ `password` (string, optional)
+ `phoneNumber` (string, optional)
+ `role` (enum[string], optional)
    + `ADMIN`
    + `PLACEMENT_OWNER`
    + `CLEANING_PROVIDER`


## AgreementRequestDto (object)


### Properties
+ `date` (string, optional)
+ `id` (number, optional)
+ `placementId` (number, optional)
+ `providerServiceId` (number, optional)


## LoginDto (object)


### Properties
+ `email` (string, optional)
+ `password` (string, optional)


## PlacementDto (object)


### Properties
+ `area` (number, optional)
+ `floor` (number, optional)
+ `id` (number, optional)
+ `lastCleaning` (string, optional)
+ `placementType` (string, optional)
+ `smartDevice` (SmartDeviceDto, optional)
+ `windowsCount` (number, optional)


## PlacementOwnerDto (object)


### Properties
+ `address` (AddressDto, optional)
+ `creationDate` (string, optional)
+ `email` (string, optional)
+ `id` (number, optional)
+ `name` (string, optional)
+ `password` (string, optional)
+ `phoneNumber` (string, optional)
+ `role` (enum[string], optional)
```

```
    + `ADMIN`
    + `PLACEMENT_OWNER`
    + `CLEANING_PROVIDER`


## ProviderServiceDto (object)


### Properties
+ `description` (string, optional)
+ `id` (number, optional)
+ `maxArea` (number, optional)
+ `minArea` (number, optional)
+ `name` (string, optional)
+ `placementType` (string, optional)
+ `pricePerMeter` (number, optional)


## Role (object)


### Properties
+ `id` (number, optional)
+ `name` (enum[string], optional)
    + `ADMIN`
    + `PLACEMENT_OWNER`
    + `CLEANING_PROVIDER`


## SmartDeviceDto (object)


### Properties
+ `adjustmentFactor` (number, optional)
+ `airQuality` (number, optional)
+ `dirtinessFactor` (number, optional)
+ `humidity` (number, optional)
+ `id` (number, optional)
+ `priority` (string, optional)
+ `temperature` (number, optional)
```