**Targets**

Setting targets:

Implement an application for scheduling and organization meetings in organization. The organization has offices, each office has meeting rooms. Employees can attend meetings in different offices.

Technology-stack:

1. UI: Thymeleaf or another UI-technologies
2. Java + Spring + additional libraries (Lombok, Liquibase, Swagger) + Testing (Junit, Mockito).
3. PostgreSQL; GitHub, Maven.

System requirements:

1. The application must work and interact according to the rules from the subject area.
2. Implement a REST application with necessary CRUD requests to interact with DB.
3. Add authorization/authentication.

Questions:

1. Determine the restrictions of the subject area (What is acceptable?).
2. Create a model of the subject area – determine the entities and their relationship.
3. Learn technologies-stack and system concepts.

**The Subject Area**

The subject area is a system with the following entities:

* Offices
  + Address;
  + Phone number
  + Inventory in the office warehouse
* Rooms in offices, where meetings take place
  + Capacity
  + Office, where the room is located
  + Inventory
* Employees
  + Name
  + Position
  + Application account
  + Email
  + Office, where the employee is working
* Meetings
  + Meeting room in the office
  + Time
  + Theme

Entity features:

* Offices can be located in different cities
* Employees can have different roles (admin, user)
* Meeting rooms can be filled with some inventory items

Entities restrictions in the section – Database.

**Application**

Description:

Web application for scheduling and organizing local meetings in offices.

User features:

* Convenient user interface
* User registration and authorization; employees can request the creation of an account from the admin
* Storing users and their meetings information
* Scheduling local meetings (Create, edit, cancel)
* Sending notifications to users

Internal structure:

* Backend: PostgreSQL, Liquibase, Java (Spring, Lombok, Swagger, Junit, Mockito)
* Frontend: Angular

**Database**

PK – mark for primary key fields in the table. Primary key must be unique.

U– mark for unique fields in the table (Not primary key).

FK – mark for foreign key fields in the table, main table in brackets.

Database structure:

* Office table
  + Office ID [PK]
  + City
  + Address
  + Phone number
  + Open time
  + Close time
* Room table
  + Room ID [PK]
  + Office ID [FK (Office table)]
    - One office can have several meeting rooms
  + Capacity
    - Can`t invite more people than the room can contain
* Room inventory table – a table for linking rooms and inventory items
  + Room ID [PK, FK (Room table)]
  + Item ID [PK, FK (Items table)]
  + Count of items in the room
* Inventory items table
  + Item ID [PK]
  + Office ID [FK (Office table)]
    - If need to request items to the room, they must be in stock at the office warehouse
  + Count of items in the office warehouse
    - Can`t request more items than are in the stock
* Employee table
  + Employee ID [PK]
  + Name
  + Position
  + Email [U]
  + Office ID [FK (Office table)]
* User data table
  + Employee ID [PK, FK (Employee table)]
  + Login [U]
  + Password
  + User activation
* Employee role table – a table for linking employees and roles
  + Employee ID [PK, FK (Employee table)]
  + Role ID [PK, FK (Role table)]
* Role table
  + Role ID [PK]
  + Role name
* Meeting table
  + Meeting ID [PK]
  + Meeting name (To announce meeting main theme)
  + Room ID [FK (Room table)]
    - Can`t place two and more meetings in one room at the same time
  + Office ID [FK (Office table)]
  + Start date and time
  + End date and time
* Invite table (Table for linking employees and meetings)
  + Employee ID [PK, FK (Employee table)]
  + Meeting ID [PK, FK (Meeting table)]
  + Invite accept
    - Employee can accept invite or refuse
    - Employee can cancel their current meeting to accept a new one

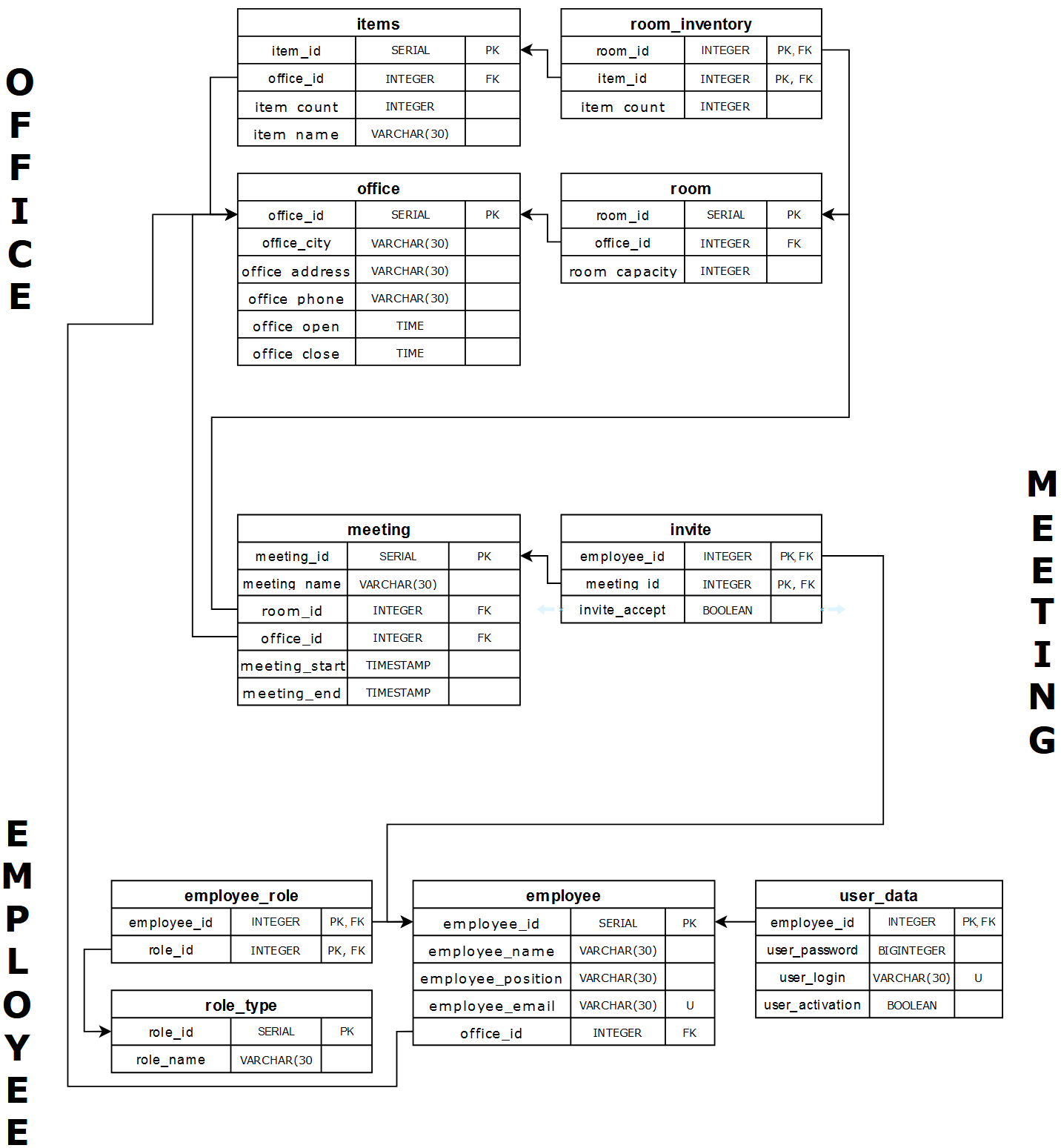


Figure 1. UML diagram for DB

**Security**

For security in the application, it is necessary to implement authorization and authentication. To achieve this goals, were used Spring Security Framework (It gives us the opportunity to implement the necessary security mechanisms in the application) and JWT tokens (It gives us the opportunity to uniquely identify users with roles).

In the application, users are divided into three groups: an admin user, a regular user and user before authorization.

Granted opportunities:

* User before authorization:
  + Login
* Regular user:
  + Creating meetings
  + Accepting or declining meetings invites
  + Change information about yourself
* Admin:
  + Full application functional

Initially, only the admin is present in the system and only he can create new users.

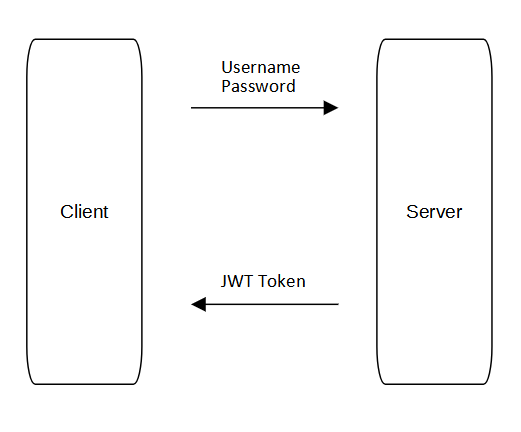


Figure 2. Security Scheme

When a user visits a website, he is taken to an authentication page where he must enter his username and password. Then, a request with his data is sent to the server, and if such a user exists, he receives back a JWT token from the server and s redirected to the home page, from where he can continue interacting with the application. After authentication, the received JWT token will be embedded in each subsequent request from client to the server to gain access to another web application functionality. Each token is valid for one hour, after the expiration of its validity, it is necessary to authentication again.