SPŠE Ječná

C - Information Technology

Ječná 30, 120 00, Praha 2

Task list app / Task list backend

(https://github.com/SasaTurtle/TaskListBackend/

https://github.com/SasaTurtle/TaskList)

Alexander Sandeep Komínek

IT

2023

Contents:

Goal of this Project:

The goal of this project is to successfully and **securely** upload and download tasks from a database into a user-friendly android app which is able to store the information for each separate user. That means registering the user and having the option to access this information through different devices and accounts.

To accomplish this goal, I had to:

a. Create a login and register user interface[1]
b. Create a Java Spring Boot API Server with PostgreSQL Database[2]
c. Use JWT (java web token) to securely access information's from the database
d. Connect the android application with the server and database

Software:

Server:

Prerequisites:

- Java version 17
- PostgreSQL
- Spring version 3

Server installation:

- https://secret-river-54565.herokuapp.com/

Server APIs:

Task API:

- POST /api/task/ - Create a new task with title, description, start date, end date, priority, and status for the current user. Requires a valid JWT token in the Authorization header.

- GET /api/task/ - Get all tasks for the current user. Requires a valid JWT token in the Authorization header.

Example:

Request:


```
[
  {
    "id":"fea62087-936f-4cd6-a581-ffa258b170a8",
    "name":"task test",
    "description":"description",
    "dateFrom":"2023-01-06T10:05:12",
    "dateTo":"2023-02-06T10:05:12",
    "status":0,
    "priority":1
  }
]
```


Response:

true

Authentication API:

- POST /api/auth/signup - Register a new user with username or email, and password. Returns a success message and a JWT token.
- POST /api/auth/signin - Login an existing user with username or email and password. Returns a success message and a JWT token.

Examples:

Request:


```
{
 "username":"ExampleUser",
 "password":"1234"
}
```


Response:

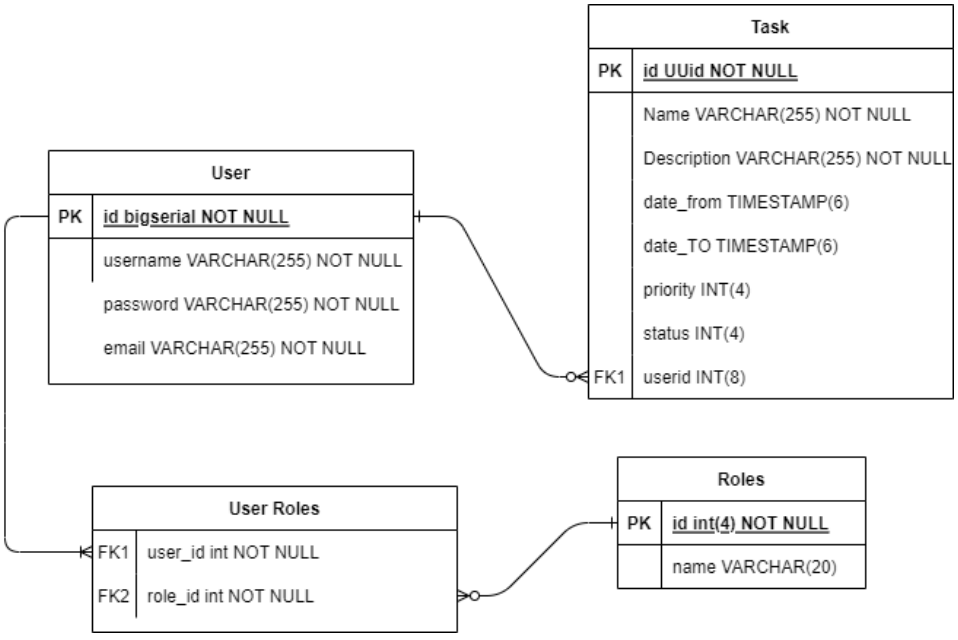

```
{
  "id": 1,
  "username": " ExampleUser ",
  "email": "example@examplemail.com",
  "roles": [
    "ROLE_USER"
  ],...

...
```

```
"accessToken":
"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJTYXNhIiwiaWF0IjoxNjg1MTAxNTIzLCJleHAiOjE2ODUxODc5MjN9.QT7xsdarvzeCRDT7B-dv6HTe4r2eh2ZqA1cgbRVg5Qd1d4iUHIc42-Z1yCAk8d6HFqJxE7x4OnOWa_0wwYu3uQ",
  "tokenType": "Bearer"
}
```


Server Architecture:

3-layer application:

1. On the firs layer are controllers providing REST API
2. The second layer secures communication between the first and third layer + business logic. The service package is responsible for this.
3. Data layer, uses JPA library and connects server with PostgreSQL database. Configuration is stored in 'application.yml' and database credentials are located inside environmental variables of the device.



Server security architecture[2]:

- WebSecurityConfig is the crux of our security implementation. It configures cors, csrf, session management, rules for protected resources. We can also extend and customize the default configuration that contains the elements below.
- UserDetailsService interface has a method to load User by *username* and returns a UserDetails object that Spring Security can use for authentication and validation.
- UserDetails contains necessary information (such as: username, password, authorities) to build an authentication object.

- UsernamePasswordAuthenticationToken gets {username, password} from login Request, AuthenticationManager will use it to authenticate a login account.
- AuthenticationManager has a DaoAuthenticationProvider (with help of UserDetailsService & PasswordEncoder) to validate UsernamePasswordAuthenticationToken object.

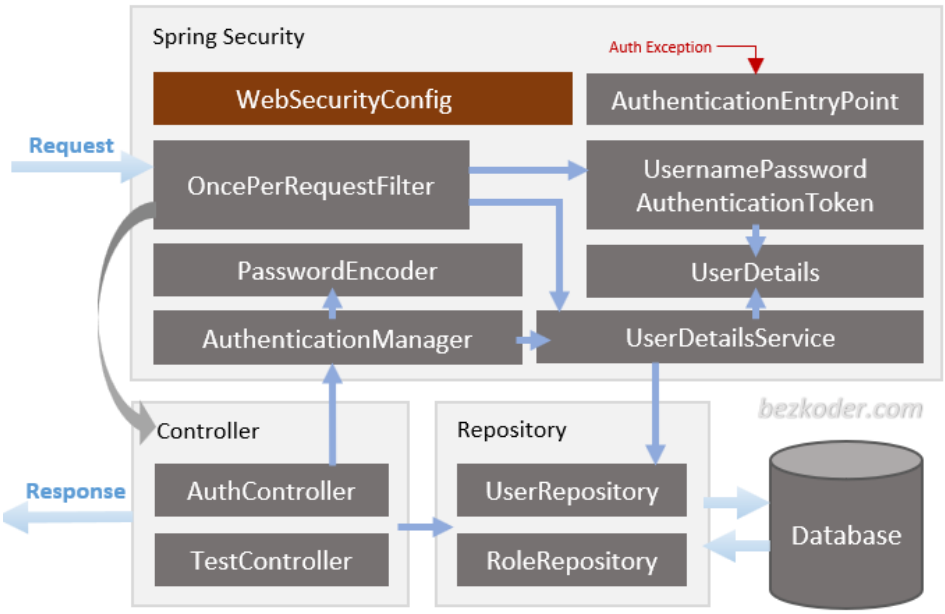
  If successful, AuthenticationManager returns a fully populated Authentication object (including granted authorities).

- OncePerRequestFilter makes a single execution for each request to our API. It provides a doFilterInternal() method that we will implement parsing & validating JWT, loading User details (using UserDetailsService), checking Authorizaion (using UsernamePasswordAuthenticationToken).
- AuthenticationEntryPoint will catch authentication error.

Repository contains UserRepository & RoleRepository to work with Database, will be imported into **Controller**.

Controller receives and handles request after it was filtered by OncePerRequestFilter.

- AuthController handles signup/login requests
- TestController has accessing protected resource methods with role-based validations.



[2]

Client:

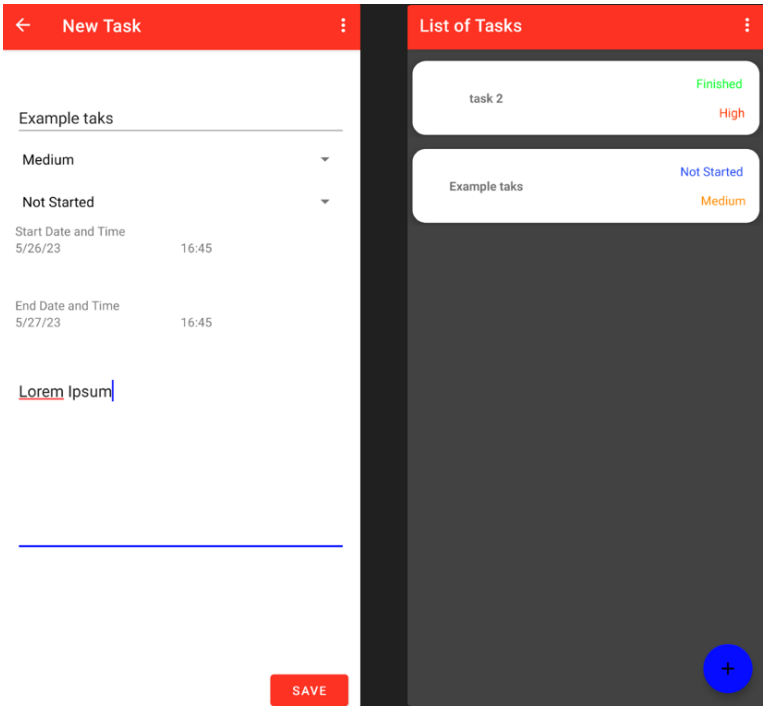
As a client I am using an android Task List application which I programed beforehand. The application only saved data locally, but I rewrote it to my current needs.

To communicate to my server, I use asynchronous Retrofit calls. To access my JWT anywhere from my project I use the SingletonToken class.

Using the app (UI):

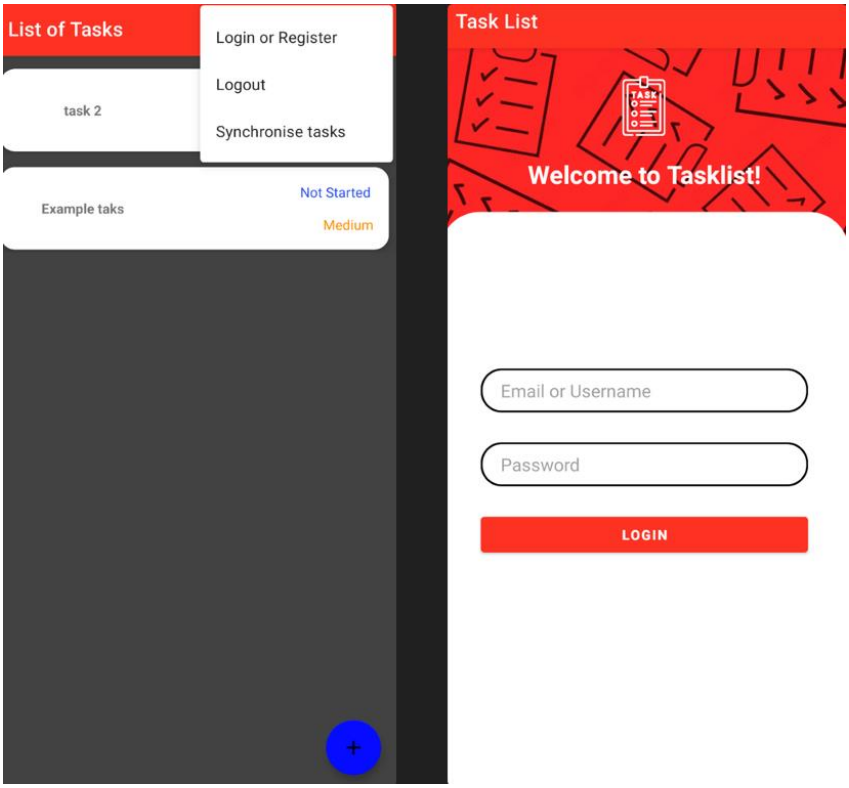
This is the main frame where the ***magic*** happens, you can view/add/update or delete tasks, login or register and synchronize with the database, all from here!

Creating, updating or deleting tasks:



To create a task, click the blue **+** button and fill out the task name, description, priority, status, start and end date time, then click **SAVE** and your task will appear on the main frame. *Voilà!*
If you wish to edit your task, then click on the task you want to edit and the same screen will appear. Change what you like and click **SAVE** again. Now your task should be changed.
After a task is deemed finished and therefore not needed you can simply delete the task by swiping from right to left on it. This action can be undone at the bottom of the screen if you are fast enough. ;)

Login, register and synchronize:



On the right upper corner of the main frame are 3 dots, this is the menu. After clicking this you will be greeted by the option to: Login or Register or to Synchronize tasks.
Login or Register: After clicking you will be relocated to the starting screen where you can either login or register by swiping left.
Synchronize tasks: After clicking all tasks associated with your account on the database will be downloaded to your device. In case of you using the app offline all tasks that were made offline will be saved to the database.

Program description:

The project is based on client – server communication. Focus in this project is the server side. The server side is based on a standard spring REST API solution which communicates with a PostgreSQL database. JPA is used for database communication. All the APIs are unit tested.

Client side is an android app which communicates asynchronously via http/https. Users are authenticated by a bearer token (JWT).

Conclusion:

Of course, there are still improvements to be made. For example, my program isn't 100% "bulletproof" and instead of the local tasks being saved in a file they could be saved in the android SQLite database. But I'm happy with what I've got.

So, in the end I'm grateful that I choose to get out of my comfort zone and to work on something new and more difficult than I'm used to. I hope to remember something and to use this knowledge in the future.

Sources:

[1]: (10) Android Login Screen | Login Android Studio | Android Studio - YouTube

[2]: Spring Boot Token based Authentication with Spring Security & JWT - BezKoder

Sources not mentioned:

https://stackoverflow.com/

Spring Boot Logging with application.yml - HowToDoInJava

Deploying Spring Boot Applications to Heroku | Heroku Dev Center (database)

https://www.tutorialworks.com/spring-boot-log-to-file/