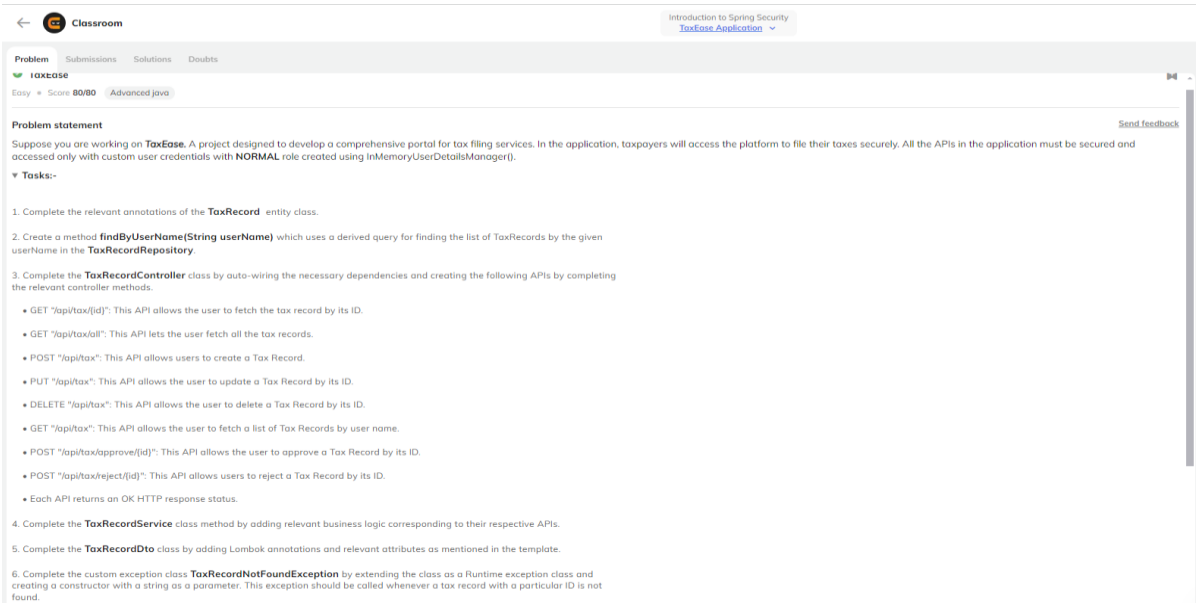
TaxEase Application

Problem Statement:


Classroom

Introduction to Spring Security
TaxEase Application

Problem Submissions Solutions Doubts

TaxEase

Easy • Score 80/80 Advanced java

Problem statement

Send feedback

Suppose you are working on **TaxEase.** A project designed to develop a comprehensive portal for tax filing services. In the application, taxpayers will access the platform to file their taxes securely. All the APIs in the application must be secured and accessed only with custom user credentials with **NORMAL** role created using InMemoryUserDetailsManager().

▾ **Tasks:-**

1. Complete the relevant annotations of the **TaxRecord** entity class.

2. Create a method **findByUserName(String userName)** which uses a derived query for finding the list of TaxRecords by the given userName in the **TaxRecordRepository**.

3. Complete the **TaxRecordController** class by auto-wiring the necessary dependencies and creating the following APIs by completing the relevant controller methods.

- GET "/api/tax/{id}": This API allows the user to fetch the tax record by its ID.
- GET "/api/tax/all": This API lets the user fetch all the tax records.
- POST "/api/tax": This API allows users to create a Tax Record.
- PUT "/api/tax": This API allows the user to update a Tax Record by its ID.
- DELETE "/api/tax": This API allows the user to delete a Tax Record by its ID.
- GET "/api/tax": This API allows the user to fetch a list of Tax Records by user name.
- POST "/api/tax/approve/{id}": This API allows the user to approve a Tax Record by its ID.
- POST "/api/tax/reject/{id}": This API allows users to reject a Tax Record by its ID.
- Each API returns an OK HTTP response status.

4. Complete the **TaxRecordService** class method by adding relevant business logic corresponding to their respective APIs.

5. Complete the **TaxRecordDto** class by adding Lombok annotations and relevant attributes as mentioned in the template.

6. Complete the custom exception class **TaxRecordNotFoundException** by extending the class as a Runtime exception class and creating a constructor with a string as a parameter. This exception should be called whenever a tax record with a particular ID is not found.

6. Complete the custom exception class **TaxRecordNotFoundException** by extending the class as a Runtime exception class and creating a constructor with a string as a parameter. This exception should be called whenever a tax record with a particular ID is not found.

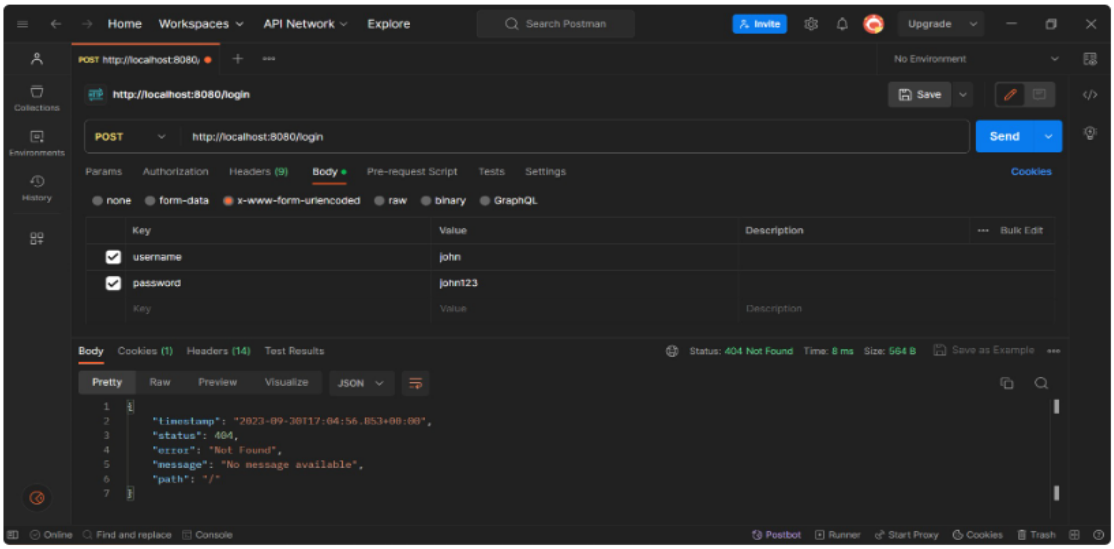
7. Complete the **TaxRecordConfig** class by creating a **"filterChain" bean**, a **"passwordEncoder" bean** and **a "users" bean**. Also, you need to add a default user with the following details:

- username: "john"
- password: "john123"
- role: "NORMAL"

8. The user bean should have a user with a role named "NORMAL".

9. Test the application using Postman as shown below:

▾ **Reference image**

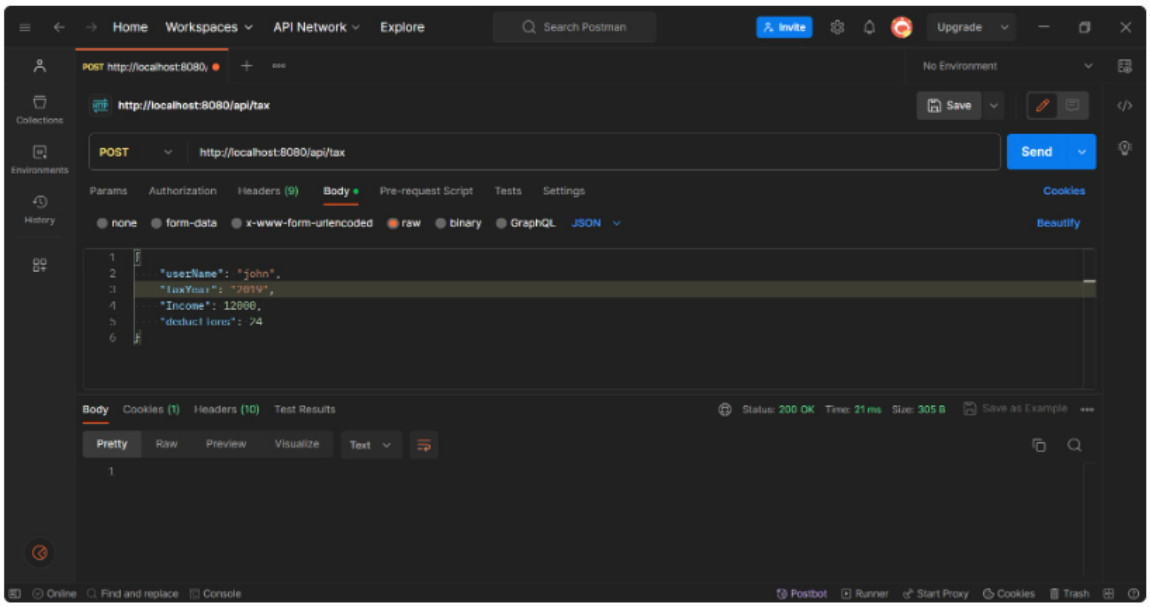


- Here we are using the "localhost:8080/login" endpoint due to Spring Security. The "/login" endpoint is added while hitting the "localhost:8080" endpoint using a browser and is managed by Spring internally by default.

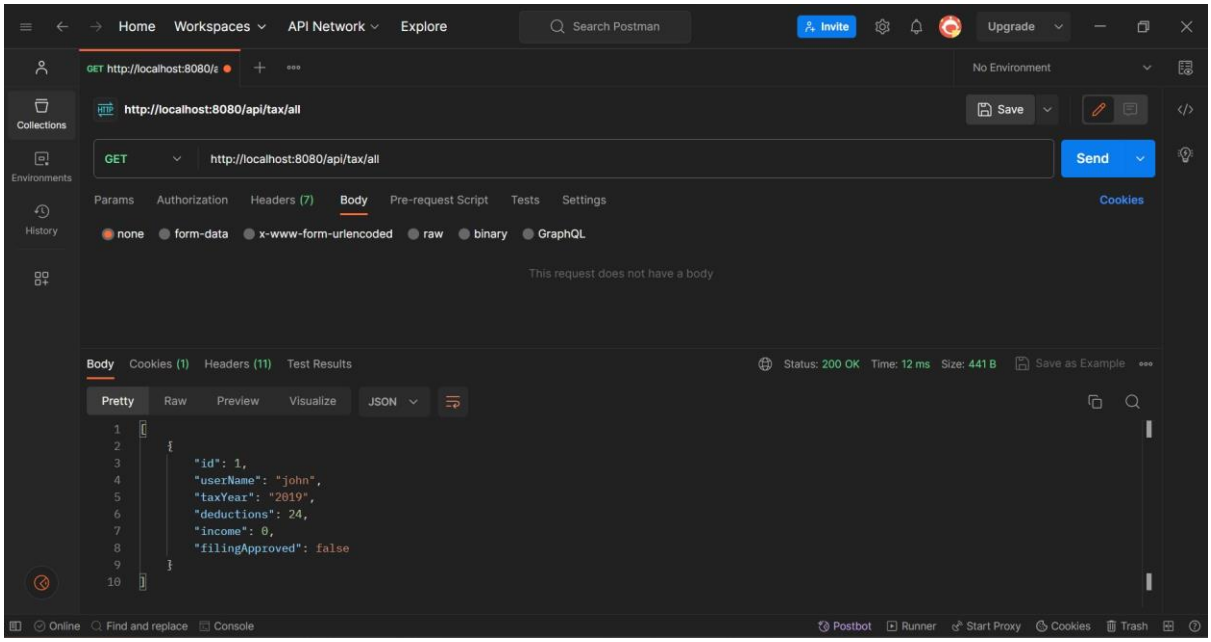
Problem | Submissions | Solutions | Doubts

• After hitting the "/login" api a session will created thus referring to a successful login. Now all the APIs can be accessed normally but in the same tab.

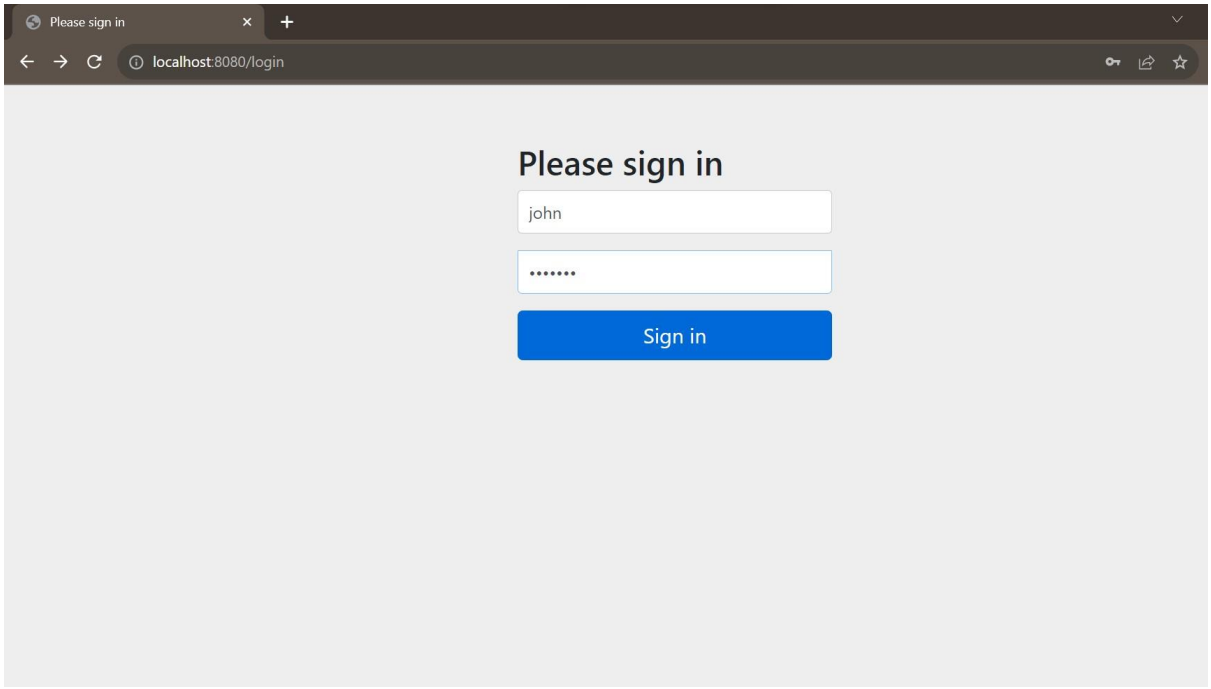
▼ Reference image



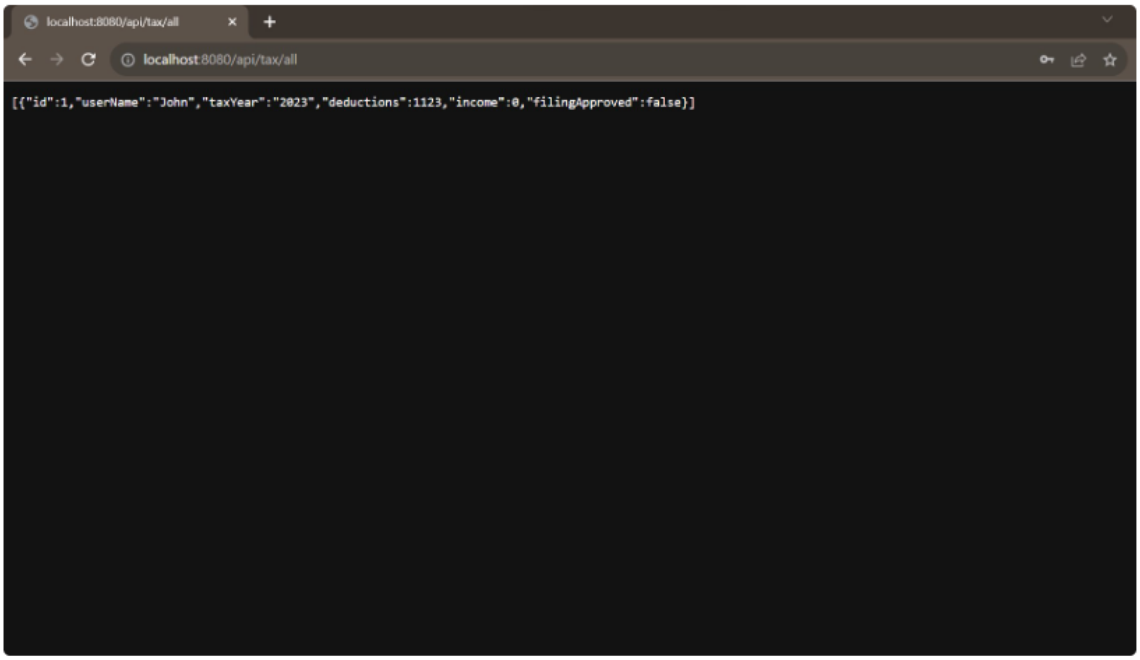
▼ Reference image



Output



• Here we are hitting the "localhost:8080" endpoint using a browser where the "/login" endpoint is added by default after hitting the API.



• After hitting the API a login form will appear provided by Spring itself. Enter your credentials and now you can validate your login by hitting a new request within the same session.

• To validate the login, we can only hit the 'GET' request through a browser.