Video Rental Store Web Service v1.0

The video rental store service described in the assignment is implemented here as a client-server fashion, divided in 4 submodules:

- vrs-server: The server module, that handles incoming requests and handles the business logic and data management. Implemented with Java8, JUnit, Spring (Boot, Data JPA, Rest) and MySQL.
- vrs-common: A module that includes DTOs and items used by both client and server.
- vrs-client: The client module written in Java and Spring, so any other service can have it as a dependency and use the video-rental-store service.
- vrs-integration-tests: A testing module that starts a mock service that uses the vrs-client to query the vrs-server and perform tests.

System setup:

> *In order to use the service your system need to have installed: Java8 JRE, Mysql 5.0+ with a root/root user and maven 3.0+. Before running the service:*

- Execute in your mysql server the video_rental_store_create.sql file.
- Go to video-rental-store folder and run: mvn clean install -DskipTests in your terminal.

Start the service:

> *Go to vrs-server folder by terminal and give: spring-boot:run . The service should start and will be waiting for requests.*

Test the service:

> *To take a good idea about how the API works, (after starting the service!) go to the vrs-integration-tests folder by terminal and give mvn test. For anyone interested to integrate the service (highly unlikely :)) you could check the integration tests, they cover most if not all scenarios. There are also unit tests in the vrs-server/src/test/java folder to get the idea of the way I implemented the business logic.*

Endpoint: http://localhost:9081/vrs/

Following are the available functions / API calls.

RentalResultDTO rentFilms(RentalListDTO rentalList)

> *The service user creates a list with the rental information of one or more films a customer is going to rent, creates the RentalDTOs (see vrs-common module for all DTOs used) and adds them on a list. The list is wrapped by a RentalListDTO that is given as an input to the call. The response consists of a RentalResultDTO, which currently just contains the initial rental charges for the customer.*

ReturnFilmsResultDTO returnFilms(ReturnFilmsDTO returnFilmsDTO)

> *The service user creates a ReturnFilmsDTO that contains a list of the ids of the returned films by the customer, customer's id and the datetime of the return. The response consists of a ReturnFilmsResultDTO, which currently just contains any rental surcharges for the customer.*

CustomerDTO createCustomer(CustomerDTO customerDTO)

> *Insert a new customer in the system. The service user creates a new customer by passing a CustomerDTO filled with a new customer's data, and the responce contains a CustomerDTO with customer's data that are persisted on the system.*

CustomerDTO updateCustomer(CustomerDTO customerDTO)

> *Update an already existing customer in the system.*

CustomerDTO findCustomer(CustomerDTO customerDTO)

> *Search for a customer in the system.*

FilmDTO createFilm(FilmDTO filmDTO)

> *Insert a new film into the system.*

FilmDTO updateFilm(FilmDTO filmDTO)

> *Update an already existing film in the system.*

FilmDTO findFilm(FilmDTO filmDTO)

> *Search for a film in the system.*

Decisions made

> *Due to the nature of the project (assignment) and the zero posibilities of its continuation, I paid no attention to the Javadoc and aimed for self explanatory code instead. For the same reason I neglected any serious exception handling (project specific business logic exceptions included) and added no version support on the API. I decided to go for Java instead of JSON API, to demonstrate both client and server handling skills in Java. I focused on clean contracts for the services and the API, on testing, both unit and integration, and while developing the server part I followed a TDD approach (write tests first, red->green->refactor). I strived to use (but not to overuse) interesting features of the language like streams and map/reduce. Overall I strived to follow a KISS approach.*

Contact Details

LinkedIn: Tasos Tsaousis Personal Email: tasoskanenas@gmail.com

P.S

> *I found the project fun to code through, and I would be gratefull for any kind of constructing feedback upon my code and my general design and approach in the subject. Thanks a lot.*