Programming II

Project 1: SuperShopManager

Super Shop Management API (20 Points)

Deadline **Sunday 09.04.2023, 23:59 CET.**

Let's imagine, that we are building up a new software for management of online shops. The software shall be used by employees of the shop to manage the inventory, customers, monitor the status of sales and similar tasks in the online shop. It will also serve as a common interface to the customers.

This programming task is about implementing a web API, so that the same backend can be customized by different shops to build their own applications based on the provided API. As of now, we do not care about the front-end of the application, which could be any browser or a mobile app. And we also don't care about a database that stores the data persistently. In this exercise, we deal with the API implementation only. You may use a **REST client, e.g., Postman** to quickly see the results of the API call. It is mandatory to use the **Python requests library** and **Pytest** to thoroughly test the API. A **Swagger based UI** is provided, so that you can also test the application based on a browser front-end. Design your objects and classes allowing for easy future extensions

The API consists of the following functionality, some have already been implemented as examples:

Management of customers

- Get a list of all customers, or a specific customer data

Type	done	URL Path	Description
GET	✓	/customer	*Return a list of all customer data*
GET	✓	/customer/<customer_id>/	*Return the data of the customer with the specified customer_id*

- Register or remove customers to/from the shop. For each customer store a customer-id, date of birth, shipping-address, email-address and bonus-points earned so far.

Type		URL Path	Description
POST	✓	/customer	*Register a new customer, fields: name, email, dob, address*
DELETE	✓	/customer/<customer_id>/	*Delete an existing customer, field: customer_id*
PUT		/customer/<customer_id>/	*Update customer's data, all fields can be updated, except the email address*

- Upon registration, the customer data is stored and flagged as "unconfirmed". The system generates a random number (verification token) and stores it. This would typically be sent to the customer's email address. The customer can use this number to verify the ownership of the email address. Only verified customers can login to the platform.

Type		URL Path	Description
PUT	✓	/customer/verify	*Verify customer's email address, fields: customer_id, verification_token, email.*

- Customers sometimes forget their password. So, the system also provides a feature to reset the password. A random temporary password is generated and sent to the user's email address. Users can use the temporary password to create a new password.

Type		URL Path	Description
POST		/customer/pwreset	*Generate a temporary password and send via email, fields: customer_id. For this exercise, emails don't really have to be sent.*
PUT		/customer/pwreset	*Allow password reset based on the temporary password, fields: customer_id, temp_pw, new_pw.*

Management of Products

- Add/remove products to/from the system. For each product store the quantity (how many items are available)

Type		URL Path	Description
POST		/product	*Add a new product. Each product has a product-id, name, serial number, expiry date and a category.*
DELETE		/product/<product-id>	*Delete an existing product, field: product_id*
PUT		/product/<product-id>	*Change the stock of an existing product, e.g., when new items are delivered to the warehouse.*

- Get details of products.

Type	done	URL Path	Description
GET		/products	*Return a list of all product data*
GET		/product/<product_id>/	*Return the data of the product with the specified product_id*

- When a product is sold, make sure it is added to the purchase history of the customer and removed from the inventory.

Type	done	URL Path	Description
PUT		/product/sell	*Sell a product, fields: customer-id of the buyer, quantity of product sold*

- Allow removal of items without being sold, e.g., damaged items or products with expiry date less than 48 hours. This is not the same as "DELETE".

Type	done	URL Path	Description
PUT		/product/remove	*Remove an item from inventory, fields: product-id, reason for removal*

- When the stock of items goes below a certain threshold (t), items must be purchased by the supermarket. The threshold is equal to the quantity of items sold in one week.

Type	done	URL Path	Description
GET		/products/reorder	*Display a list of all products, that must be reordered this week.*

Management of customer orders

- Customers have a shopping cart, to which they can add products they wish to buy. Products can be added/removed to this cart, quantities can be changed etc.

Type	done	URL Path	Description
PUT		/customer/<customer_id>/add2cart	*Add an item to shopping cart, fields: product-id, quantity.* To delete an item from cart, set the quantity to -1.

- Order is confirmed only when the customer provides valid credit card numbers. Use any "mock credit card verification" system to simulate this service.

Type	done	URL Path	Description
POST		/customer/<customer_id>/order	*Confirm an order. Fields: shipping address, credit card number. For every euro the customer spends, she gets one bonus point. Customers can pay the for the items using their customer bonus points. Each bonus point is equivalent to 10 cents during payment.*
GET		/customer/<customer_id>/orders	*For each customer order, keep track of when the order was placed, when the product was delivered. Return this list with this API call.*
GET		/customer/<customer_id>/returnable	*The law in Austria requires that online shops must allow customers to return online purchases within two weeks of purchase. Keep track of which products could still be returned. Return a list of such items for the specified customer.*

- Each customer has a purchase history, which can be used to recommend new products to the customer.

Type	done	URL Path	Description
GET		/customer/<customer_id>/recommendations	*Get a list of 10 best products to be recommended based on the customer's purchase history.*

- For every euro the customer spends, she gets one bonus point. Bonus points can be used to buy items in special offers.

Type	done	URL Path	Description
GET		/customer/<customer_id>/points	*Return the total bonus points earned by the customer so far.*
PUT		/customer/<customer_id>/points	*Add bonus points. Sometimes customers get extra bonus points, e.g., on their birthdays. These are added to the total bonus points.*

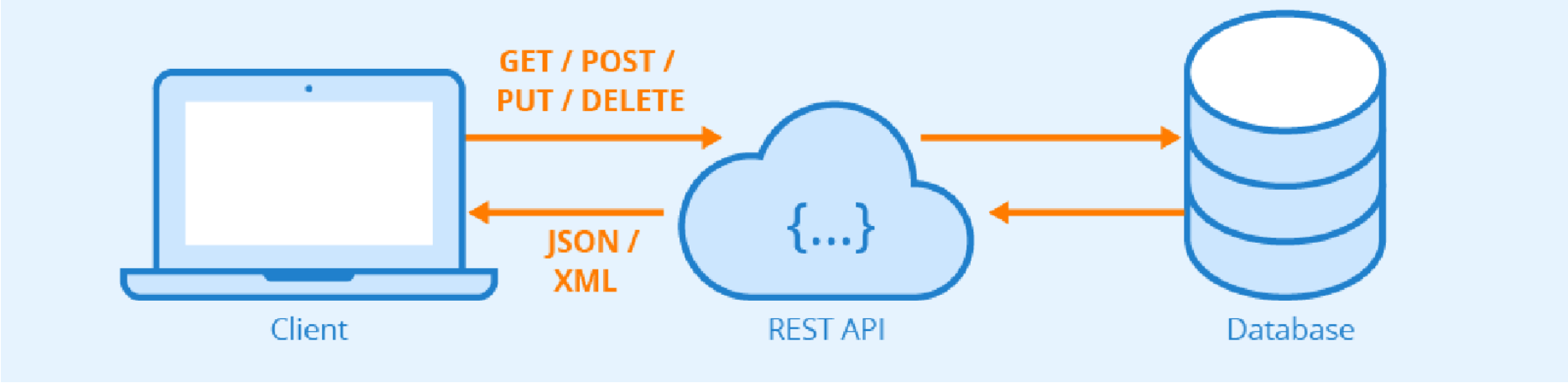
Management of festival and weekend sales discounts

- The shop management may decide to have special discounts (valid for a few days) on certain categories of items. These apply to all order placed in the timeframe of validity of the discounts.

Type	done	URL Path	Description
POST		/coupons	*Add a new coupon (a 10 digit number) for a product category, specify the validity dates for the coupon and the discount percentage for the products in the specified category.*
GET		/coupons	*Get a list of all currently valid coupons.*

The API is implemented in Python using a package called Flask, which allows you to define HTML methods GET, POST, PUT etc. Each method returns a JSON object, which can be used by the front-end application in adequate ways. The summary of HTML methods to be implemented as part of the homework is listed in the following table. The first few methods have already been implemented as **examples**.

Write **Test cases to thoroughly** test your API – manual testing with Postman is good, but **not enough.** Come up with an automated testing script, which simulates the daily operations of the shop. For example, add a few customers, add discounts and generate coupons, simulate shopping behavior, damaged products, etc.



In this exercise, we focus on the API implementation only. Feel free to create your own client application based on the API as your side project. The side project is not graded but a lot of fun!

Example code implementing the first five methods is provided in a Github repository.

https://github.com/deepak-dhungana/INF-SS23-Programming-II

Submission of the project must be done through a Github Repository. Create your own repository and upload your code to your private repository before the deadline. Add https://github.com/deepak-dhungana as a collaborator to your repository. Submit the URL of your repository via MS Teams.

In order to create a new Github account, visit https://github.fhkre.ms and login with your IMC FH Krems account.