Task 1: A web shop with a REST API

The task is to implement a basic web shop without a payment solution using a RESTful API (except that). You don't have to adhere to all the HATEOAS requirements, specifically your JSON results do not have to contain links.

Note that this course is not a web design course and especially for smaller groups you're not expected to make anything that looks fancy. There is no need for good looking CSS - simple default HTML look & feel is ok. That said, you're free to use CSS frameworks like bootstrap and JavaScript frameworks such as react or angular if you prefer.

User stories

As a user:

- I can see the store front page with a list of products, each with pictures or i cons. (You can borrow the pictures or make your own.)
- I can click on a product to get more information about it and possibly bigger and more pictures.
- I can see what each product costs
- It's easy to see where to click if I want to buy the product. If I click there I can see that the number of products in my "shopping cart" increased
- I can easily see how to proceed to checkout or to complete my purchases.
- When I get to the checkout I'll be shown a list of all the products I've bought and a total price at the bottom. And then, by pure luck, I'll see that a special bonus applies to me right now that deducts the whole amount allowing me to get all those products for free.

As an administrator:

If you're a small group (1 or 2) the admin functionality can be considered a stretch goal.

- If I authenticate myself with a special administrator username and password provided in the documentation I can see that I get logged i n with admin rights
- As an administrator I can add new products. Each product has at l east a name, a short description and a longer description and a photo.
- If I don't add a photo a generic placeholder photo will be used instead.

Technology choices

Backend

1. **Python3** and Flask. Tutorial here.
2. **Node.js** and the express framework. Tutorial here.
3. **Java** with spring.

Feel free to use swagger to specify and document your REST API and to generate your stubs. Note however that the generated code can be harder to read and may require some time to figure out how to use.

Database

1. **MySQL**. Probably the most deployed database i n the world. Recommended because it should be familiar to most students already.

2. **MongoDB**. NoSQL database effectively storing JSON directly as "objects" instead of rows and columns (it's really a binary j son format). Instead of a custom query language it uses JavaScript for queries. Recommended especially for node.js setups because it gives you JavaScript/JSON all the way.
3. **PostgreSQL** is allowed, and possibly others (ask first), but it may be harder for us to help you out as we haven't used all of them.

Frontend

When building against a REST API you have to use JavaScript to fetch resources. While CSS is not required it will make your life easier. Bootstrap

1. MDBootstrap and plain JavaScript, based on f etch.
2. Plain JavaScript using f etch and plain HTML.
3. MDBootstrap and j query, using ajax.
4. React.js
5. Angular.js

Helper tools

- The Postman client for testing your API using a GUI while developing it
- HTTPie for testing your API using the terminal while developing it

Stretch goals

Devops

- Secure all API endpoints with TLS (sometimes referred to as SSL/TLS).
- Add monitoring using Prometheus to track the resource usage of your service
- Add proper authentication using OAuth2 and/or a third party authentication service such as Google Sign-in

Extended functionality

- I can register with my username and email and then login.
- If I'm logged i n when I make a purchase, I can see a list of my orders on my user page.

Deployment with Docker

There must be at least two dockerfiles - one for the database and one for your backend. You should use docker compose to integrate the two. For smaller groups and groups with lower ambitions using docker compose can be considered a stretch goal.