PizzaStore

1. Preface

In Nortal, some developers eat a lot of pizza. Naturally, it makes sense to start writing a full-stack application for managing our own store (in case a client owning a pizzeria walks in and offers a discount on pizzas).

2. Tech guide

2.1 Gradle

Gradle commands can be executed in several different ways:

- When working on Windows command line or PowerShell: *gradlew.bat <command>*
- When working on MacOS, Linux or Windows 10 Shell: *./gradlew <command>*

2.2 Docker

If you do not have Docker installed and you are not familiar with Docker, then head over to https://docs.docker.com/get-started/ and install the latest version of Docker.

2.3 Annotation processor

IntelliJ IDEA:

- Install the Lombok plugin.
- Enable the annotation processor:

File->Other Settings->Default Settings
Expand Build, Execution, Deployment
Expand Compiler
In Annotation Processors check Enable annotation processing

For other IDE's please consult your IDE manual.

2.3 Assignment setup

To start the backend application, run "*gradlew bootRun*" in the pizzastore-api project directory.

The Application uses a H2 in-memory database by default. To browse the database, use http://localhost:8080/h2-console (JDBC URL: jdbc:h2:mem:testdb / user: sa)

To start the front-end application, in the pizzastore-ui directory, first run "*npm install*" and then "*npm start*" to start the application.

3. Assignment

The PizzaStore application manages two different roles: customers and admins. These two roles have different permissions within the application. Customers are able to create a new order and view their own orders whereas admins can see all the orders and also have permission to cancel them.

Existing users:

Admin -	username: *admin*	password: *admin*
Customer -	username: *customer*	password: *customer*

3.1. Tasks

At first familiarize yourself with the PizzaStore application and its code. There are some things missing from the application that have to be implemented: look for TODOs in the code. Start by reading the *CHANGELOG.md* file located in this assignment folder.

1) Bacon topping is missing from the pizza toppings

    Add a new bacon topping to the database and display the option in the UI.

    Existing image for the topping can be found in */src/assets/toppings*.

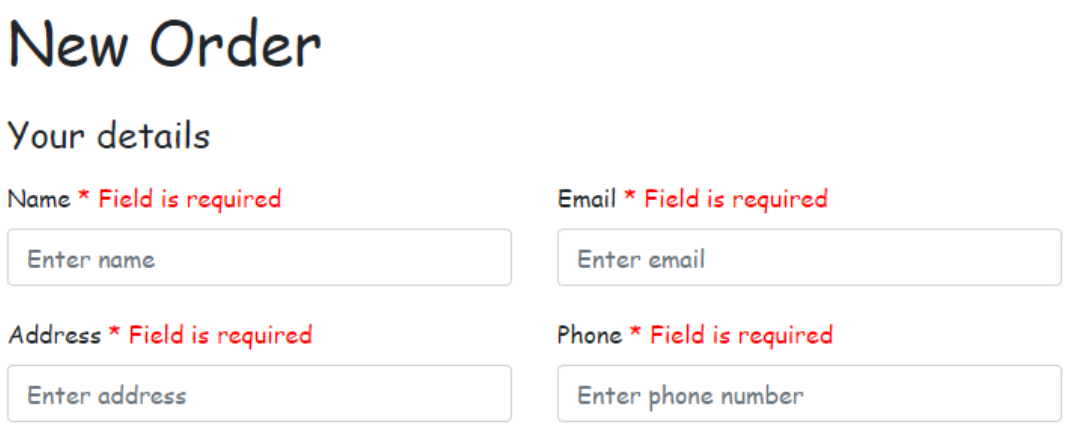
2) Customer's details are not validated when an order is placed

Add following validations for customer details form with appropriate error messages:

- Name: required
- Address: required
- Phone: required
- Email: required and matches a valid email pattern

NB! The validations must be done both on the client and server side.

Example:



3) Users cannot see the total price of each order in ORDERS/MY ORDERS view

Calculate the total price of the customer's order on the server side.

Total price of the order should then appear like this:

4) Order cancelling in admin's ORDERS view is not working properly on the server side

Cancelling an order currently deletes it from the database, but we would actually like to keep the order for audit purposes. Implement logical deletion by finding a way to deactivate orders instead of physically deleting them. Behaviour in the UI must remain the same: cancelled orders must not be shown.

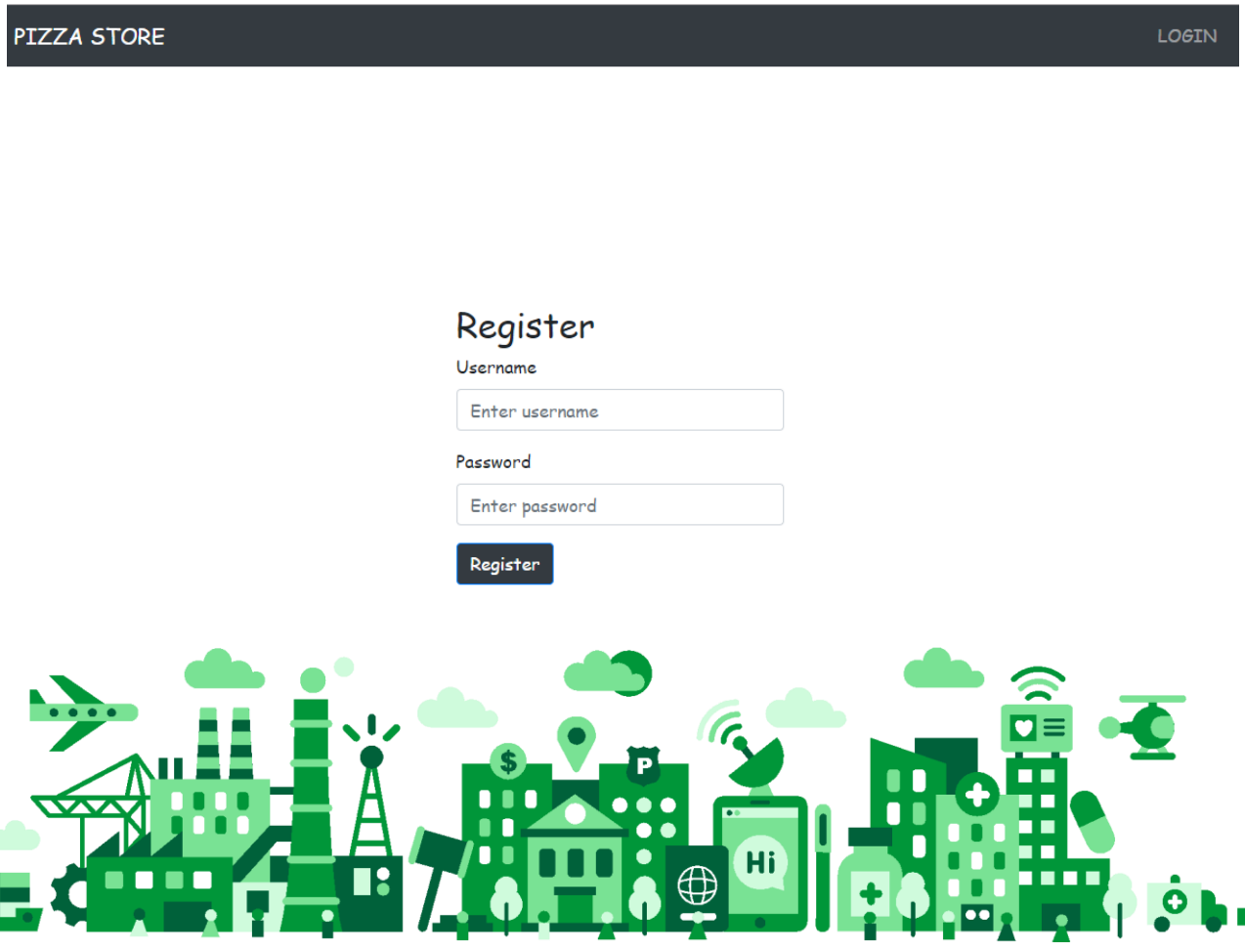
5) Currently customers do not have the option to register

Add a registration form for customers with the following requirements:

- Form has username and password input fields, both are required.
- Form has validation for required fields with appropriate error messages.
- Application checks whether username already exists or not.
- The customer is redirected to "NEW ORDER" page after successful registration.
- When registration fails, error message is displayed to the customer.
- Passwords are currently stored in plain text. Implement password encoding with a commonly known secure algorithm of your choice. NB! Encoding must be done on the server side.

Existing registration API endpoint is */auth/register*.

Example:



6) Production-grade database

H2 is good enough for development but to get production ready, we definitely need a proper persistent database. Let's set up a PostgreSQL database in Docker:

- Create a docker-compose file for a PostgreSQL database container setup. An empty file is already provided in the project root directory.

A separate Spring configuration has been provided at *pizza-main\src\main\resources\application-pg.yaml*. Use the same database settings in your docker-compose if possible. If not, then change the Spring configuration accordingly.

When the docker-compose file is ready, run "*docker-compose up*" in the project root.
When the database has started, run the application with PostgreSQL profile: *gradlew bootRun -Pprofiles=pg*

Now we have a proper persistent production-grade database.