Project Assignment : EatsDelievery

13 Jul 2021

Gonçalo Passos [88864]
João Vasconcelos [89022]
Luís Valentim [93989]
Marta Ferreira [88830]
Rita Ferrolho [88822]

Introduction

Concept

A system for food delivery, through client, courier, restaurant communication capabilities, where a client files an order based on items presented by the restaurants in their personal pages, the order is forwarded to the respective restaurant and an available courier which will later deliver the food to the client in the same vein as UberEats. However based on our own fully independent Courier/Order API which could be applied to any other courier service.

Current Solution

Due to time constraints and group management problems unfortunately we couldn't finish everything we had planned, we have simple implementations for both API's to do every basic functionality however we could not deploy or link them together for a complete final product. We still tried, to a moderate degree of success, to follow Test Driven Development and provide good verification of each of both systems functions but for the same reasons it is not as thorough as we had initially planned.

Project Specification

Functional Requirements

The system would require sign in and login functionality for users, the ability to access a restaurant page and within said restaurant page add the available products to their order as well as to remove certain items from the current order, once completed the user should be able to check the order's status, accepted, picked up and delivered. It should be able to give an order, find an available close-by courier and relay the order to him and to the restaurant. Restaurant users should be able to add and remove items and also change the information of each one,

and should also be capable of showing whether or not their services are available at the moment.

Non-Functional Requirements

As a food delivery app it's very important for our product to have high availability, maintainability and to assure each user's data privacy, and because it's heavily catered for the general public, it should be extremely easy to learn and use, each functionality has to be clearly visible and the user should be provided tips and instructions whenever relevant.

Personas and User Stories

Tiago Albuquerque

Tiago is 34 years old and works as a salesman in a large company, with a heavy workload and an inflexible schedule. Due to this, he doesn't always have the time or energy for cooking. Besides, he doesn't have an interest in cooking, so he frequently gets food through home deliveries or take aways from offices at his company.

Motivation: Tiago would like to have an easier and practical way to eat without leaving home or his office. He also intends to not be limited to the restaurants with take away or home delivery services.

Joana Carvalho:

Joana is a 23 years old student who works to help with her expenses. She enjoys walking across the city and has good social skills. As a student worker, it is not always easy to balance her studies with the work schedule, particularly during the exams season.

Motivation: Joana would like to be able to work with greater flexibility in her schedule, without being constrained to her office.

Lurdes Sobral :

Lurdes has accomplished her dream of opening her restaurant, known as “O Tasquinho”. Despite usually having many customers in her restaurant, it also handles a takeaway service. At this moment, Lurdes is considering expanding her restaurant to include home deliveries, but she does not have enough knowledge about logistics to know how to handle such service.

Motivation: Lurdes would like to include home deliveries without having to learn about the logistics of the matter.

1. Tiago is looking for the restaurant “McDonald’s”. He selects the restaurant page and selects a cheeseburger, adds it to the cart and finishes the purchase.

2. Tiago selects the cart and selects the button “remove” next to the cheeseburger product, in order to remove the product from the cart.
3. After selecting the cheeseburger from the “McDonald’s” restaurant,Tiago searches for the restaurant “Ramona” and selects a hamburger. In the menu, she selects the option “create new order”.
4. Lurdes, the owner of the restaurant “O Tasquinho”, selects “add product”, fills the form with the information of the product and selects “ok”.
5. Lurdes, the owner of the restaurant “O Tasquinho”, searches for a product, selects “edit” and fills the form with the information that pretends to edit and selects “Ok”.
6. Joana receives a delivery recommendation and selects “accept” to accept the delivery. Then, she receives the localization of the restaurant and the client location. After delivering the product to the client, she selects “finish delivery”. Since she has finished the delivery, she is now available for new delivery recommendations.

7. A user signs in to the platform.
8. A user does login.
9. A user checks which restaurants are near them.

10. A user chooses the restaurant where they intend to do the order.
11. A user creates the order.
12. The nearest available delivery man receives the order and decides to accept.
13. The delivery man delivers the order to the customer.

System Architecture

The architecture of this product [1] consists of, overall, two webapps - one for the delivery man and another for restaurant owners and regular clients. Both webapps share the same relational database. The model of the webapp used by delivery men contains the operations included in the model used by restaurant owners and regular clients. It also contains features that are only specified for delivery men.

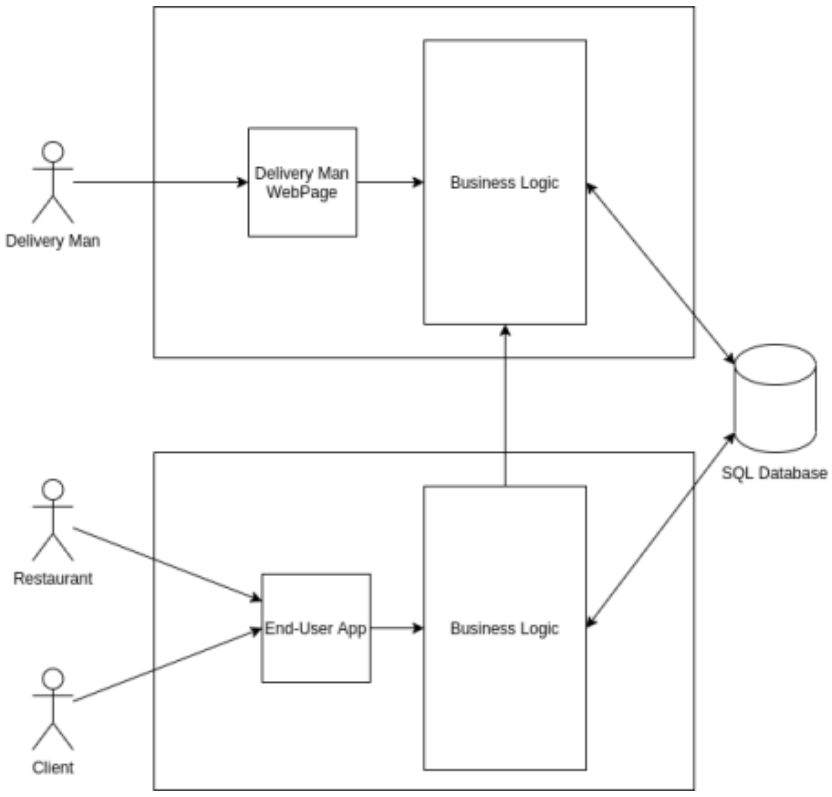


Figure 1 - Architecture Diagram

Quality Assurance Plan

In order to have tight quality assurance measures, an important aspect of this and any project, we applied agile methods following the scrum methodology with a kaban table for backlog management and feature branch workflow within our git repository, due to pressure from other projects being developed by the same team within the same time frame unfortunately we could not follow a strict scrum plan with well defined sprints, which would have surely improved code revision quality and overall quality of the product but otherwise we tried to follow as closely as possible the agile methodology as it synergizes very well with the Test Driven Development and test strategies we assumed for quality control. As for said test strategy we include unit testing of each relevant module, integration tests to make sure these interact as expected, and system and user acceptance testing to validate the system and verify if both functional and non-functional requirements were achieved. Further exploration of said tests can be found within the Quality Assurance Metrics section of this report.

Implementation

Couriers API

We used a basic implementation focused on versatility, and reusability in different scenarios, the API is supposed to work as the foundation for any courier service app, with that in mind there's two main entities to handle, the couriers and the orders for said couriers. The order entity is open enough to support any kind of service, while the courier part should handle how the couriers themselves get to operate within the API.

Couriers

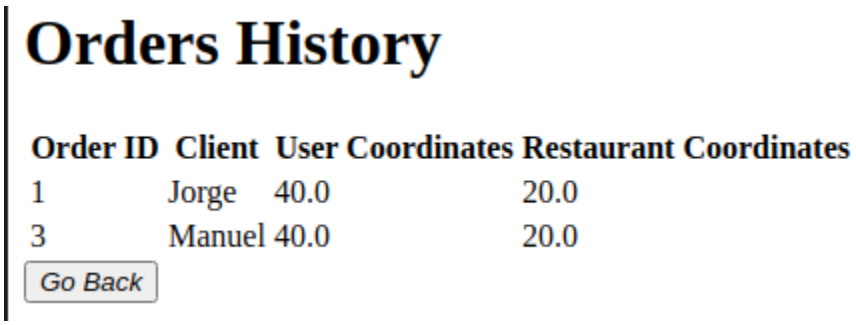
GET /api/courier/Couriers - Lists all the couriers available.

GET /api/courier/home/{courier_id} - Retrieves a list with all active orders.

Example for /api/courier/home/{courier_id}:

GET /api/courier/orders/{courier_id} - Retrieves a list with the orders of a particular courier with {courier_id}.

Example for /api/courier/home/{courier_id}:



GET /api/courier/GetOrder/{gpsCoordinates}/{courierId} - Receive the best possible order based on distance to both client and destination. In this process there's also a linkage between courier and order through {courierId}.

GET /api/courier/register_courier - Courier register page.

Couriers register themself if they are new to the service.

POST /api/courier/register_courier - Gets a valid Courier profile and registers it within the database.

GET /api/courier - Courier Login page.

Initial page where couriers log in. If they are not registered they click on the Register option.

POST /api/courier/login_courier - Courier Logs in.

DELETE /api/courier/Delete/{courierId} - Deletes Courier with {courierId} from the database.

DELETE /api/courier/Edit/{courierId} - Updates name and/or email on Courier with {courierId}.

Orders

POST /api/order/AddOrder - Receives an order and adds it to the current orders.

GET /api/order/GetOrders - Retrieves a list with all the orders.

Example for /api/order/GetOrders:

[{"id":1,"courier_id":12,"active":false,"client":"Jorge","coordinatesPickUp":20.0,"coordinatesClient":40.0,"pickUp_Name":"KFC

Glicínias"},{"id":3,"courier_id":12,"active":false,"client":"Manuel","coordinatesPickUp":20.0,"coordinatesClient":40.0,"pickUp_Name":"McDonalds Aveiro"}]

Restaurants API

An API for users to order from their favourite restaurants, focused on user experience and a small learning curve. Restaurants are able to add whatever products they have in stock and display them to their clients who can easily pick and add them to their order. Clients have the option to search for a specific restaurant and within a restaurant for a certain item of choice.

GET / - Login page.

Initial page where user if registered logs in, otherwise he clicks on the Register option.

GET /register_client - Register page

New user registers himself.

POST /register_client - Gets a valid Client profile and registers it within the database.

POST /login_client - Gets credentials, verifies them and if they're correct redirects to the home page.

GET /{username}/EatsDelivery/home - Home page.

After logging in, users can choose whatever restaurant they want to order from.

GET /{username}/EatsDelivery/home/{keyword} - Search restaurant by {keyword}.

Users use the search bar to search for their favourite restaurant.

GET /{username}/EatsDelivery/home/restaurants/{restaurant_id} - Gives all available products for a restaurant with {restaurant_id}.

After choosing the restaurant, the user chooses which products he wants and add them to cart

GET /{username}/EatsDelivery/home/restaurants/{restaurant_id}/{keyword} - Searches for a specific item {keyword} within a certain restaurant with {restaurant_id}.

Users use the search bar to search for products and add them to cart..

GET /{username}/EatsDelivery/user - User active orders and orders history.

On this page users can check all orders he has done and the ones that are not delivered yet.

Deployment

Unfortunately due to time constraints we could not deploy the system, which should have been done by deploying both API's with the respective long term storage solutions.

Quality Assurance Metrics

Unit and Integration Tests

For the unit and integration tests due to time constraints we decided to focus on the courier app since it's a more technical app which needs to have a very robust API in order to be adaptable since an important factor is trust that the system has been verified properly.

Each Service and Controller were tested to ensure their functions were behaving as we expected them to, both in isolation and integrated with other elements following an Arranje, Act, Assert technique. We could not get all the tests to pass in time for this delivery, tests still producing errors have been commented in order for the app to be runnable.

System Tests

For system testing, we used Selenium IDE for both APIs but they failed the moment it encountered a button that is part of a Thymeleaf for each. In the end, only the register and login were tested.

Functional Tests

For functional testing, we talked to a 50 years old man and a 21 years old woman and proposed a few tasks to them. The first one was on the Estafetas' API, where they had to register a new account and accept an order. Both of them had no problems doing it. The next one was to see their orders history which was even more simple, so they did it easily. On EatsDelivery we proposed two tasks as well: register a new account and add two cheeseburgers and a Big Tasty from McDonalds to cart and use the search bar to search both the restaurant and the hamburgers. Once again they did it without any problem. In the end we asked them how they felt using both platforms and they said that they are easy and intuitive to use since they are very simple.

Static Code Analysis

We tried to user sonarqube however it was initially setup within the pipeline which we couldn't fix in time and later when we tried to run it manually it turned out our licenses had expired and we couldn't get new ones in time. Other solutions could have been used however due to lack of time we decided to prioritize testing and having a functional demo.

Pipelines

Since the early days of the project we had github actions configured for a simple CI pipeline which ran sonarqube which would both run static code analysis and assure the project build was stable with clean verify, however we were not familiar with this tool and due to time

constraints we could not fix it in time, due to the rushed nature of development due to our lack of good time management the pipeline wouldn't be as effective, we were aware of this and as such didn't prioritize the pipeline.

We did not reach any deployment solution thus no continuous deployment was added to the pipeline.

Conclusion

This has been a project heavely plagged by group management issues and lack of time. However we tried to focus on having a functional demonstrable project with the base requisites and with a decent coverage of different tests. We realize our result isn't what was expected of a 5 member team in this span of time, we had several problems with the same group in multiple projects which accumulated work and unfortunately this project suffered the most. We still managed to build functional API's for a generic courier system and our specific product, although not perfect and not completely verified.

References

[1] https://www.thymeleaf.org/doc/tutorials/2.1/thymeleafspring.html - Thymeleaf documentation
[2] https://getbootstrap.com/docs/5.0/getting-started/introduction/ - Bootstrap
[3] https://docs.github.com/en/actions - GitHub Actions Documentation
[4] https://www.baeldung.com/spring-jdbc-jdbctemplate - Spring JBDC
[5] https://www.browserstack.com/guide/setup-qa-process - Quality Assurance Planning
[6] https://www.guru99.com/all-about-quality-assurance.html - Quality Assurance Planning
[7] All materials available on eLearning - Quality Assurance Planning