TQS: Product specification report

Martinho Tavares [98262], Tomás Candeias [89123], Afonso Boto [89285], Rodrigo Lima [98475]
v2022-05-13

1 Introduction

This document has the goal to capture the main project requirements and considerations. It tries to go into detail about every aspect of the project.
Every change to the document must be consented to by the consensus of the working project group.

1.1 Overview of the project

This project tries to provide a delivery service for medium to small companies, by providing ease of use and responsive tracking. While providing an example proof of concept complementary web application to understand better the client-side features provided.

1.2 Limitations

Deliveries platform:

- No delivery location tracking, only of its state
- No business-related metrics for the Admin dashboard
- Not all operations are thoroughly authenticated
- Dynamic API documentation (with Swagger) conflicted with tests and wasn't implemented

Business/store (LaundryAtHome):

- No admin view for the web app, only for the mobile app
- Webapp doesn't have a delivery location on some kind of Map, only the mobile app has
- A user can't view a complaint made in the past
- A user can't cancel an order, even if the rider wasn't found yet

2 Product concept

2.1 Vision statement

The project is composed of two sub-projects. The first sub-project is called Qourier and it's being built with the goal of bridging the independent everyday courier and big companies who need something to be delivered on a small scale.

The project will have four key deliverables:

1. Prototype project focused on UX and story-driven (26/5)
2. Core stories implementation and CI integration to the backend (02/6)
3. API final implementation, CD pipeline integration, and QA Manual final version (09/6)
4. Stable MVP, deployment of project on the cloud, final product specifications report (17/6)

2.2 Personas

The sub-project Quorier will have the following personas:

1. Alberto (administrator): manager of the overall system, who will be responsible for managing the other accounts' lifecycle and accepting registrations. They also have access to pages for monitoring the deliveries in progress/done and the platform's status.
2. Diego (rider): responsible for doing deliveries requested by the concrete delivery services. They have access to a profile with registration details and a section of basic statistics on their performance. All delivery operations, such as accepting and completing a delivery, are done in the respective section
3. Cristina (customer/concrete delivery service): the service that is integrated into the delivery engine. They can apply for a registration process similar to the rider's, and have access to their profile as well, which includes an API key for programmatically doing operations. All deliveries requested for the service can be checked in the respective section, where new deliveries can also be created

The sub-project LaundryAtHome will have the following personas:

- Jonas(Customer)
- Sofia (Customer)

2.3 Main scenarios

We present below all defined user stories for each persona of each sub-project. For each story, the acceptance criteria are listed below.

Qourier

Diego user stories:

1. Bid a delivery job
    - Visual feedback with "activated" theming;
    - Registered the Rider's bid in the database;
2. Accept a delivery job
    - Bid operations restricted;
    - Visual feedback with "accepted" message;
    - Registered as the Rider for the job in the database;
    - Delivery job not up for bidding;
3. Confirm a delivery job
    - Registered the delivery job as done;
4. Check the profile and stats
    - Is on the profile page, with the details inputted at registration;
    - Contains a section with statistics about, at least, the number of deliveries done and average time spent;

Cristina user stories:

1. Register delivery
    - Delivery up for bidding;
2. Check the progress of deliveries
    - Contains a table with all deliveries requested, in progress, and completed;
    - Deliveries in progress contain a progress bar;
3. Check the profile and stats
    - Is on the profile page, with the details inputted at registration;
    - Contains a section with statistics about, at least, the number of deliveries requested and time taken until delivery completion;
    - Contains the API key for the operations, which is hidden by default;

Alberto user stories:

1. Check Riders' progress
    - Has a table with all riders that accepted delivery and their progress, with a progress bar
2. Check each Rider's performance
    - The Rider has a section with statistics, such as the number of deliveries done and average time spent
3. Check each Rider's profile
    - The Rider has a profile with all details inputted in their registration
4. Check each Customer's stats
    - The Customer has a section with statistics, such as the number of deliveries requested and time taken until delivery completion

5. Check each Customer's profile
    - The Customer has a profile with all details inputted in their registration
6. Accept/Refuse Rider/Customer applications
    - There is an Applications page with two tables, one for Riders and another for Customers, where each row is a concrete application
    - Clicking on an application summons a modal with details of the application, with two buttons from which the application can be accepted/refused
    - Contains per-table filtering options for, at least, Accepted or Refused applications. Only accepted applications are shown by default
7. Suspend/Reactivate accounts of Riders/Customers
    - Within the Rider/Customer profile, there should be a button to suspend/activate their account
    - Suspending an account disables all operations from the account
    - Activating an account enables all operations from the account
8. Check platform performance and the number of API requests
    - Check platform performance metrics in graphs
    - Check the rate of API requests in graphs

LaundryAtHome

Jonas' User Story

Background:

- He doesn't have a laundry room at home.
- He has colored shirts for laundry

1. Request Laundry Service
    - He selects the service "wash and laundry".
    - He chooses the type of item, and finally
    - He chooses the color whether it is white or colored
    - He chooses the number of items that correspond to the description.
    - He clicks make order

Sofia's user story

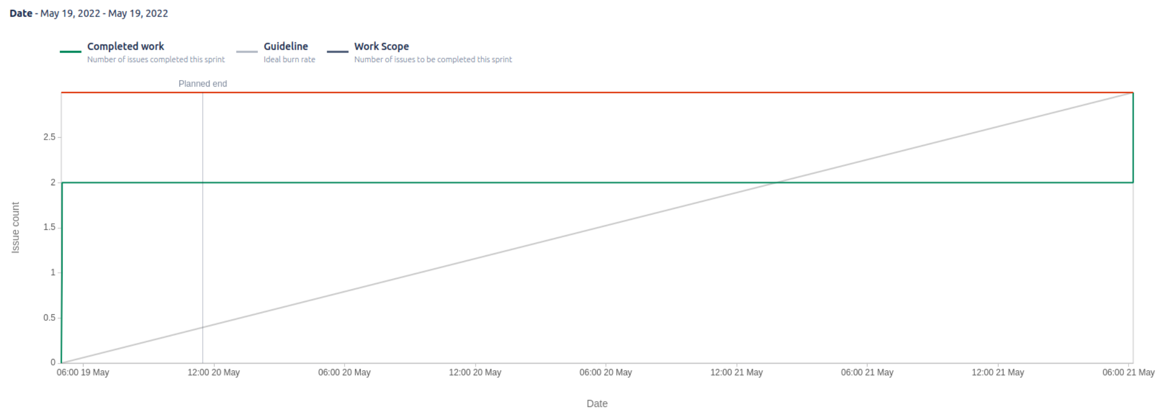
Background:

- She is a recurrent customer
- She made a couple of orders a few days ago
- She wants to see the status of an order made to her home

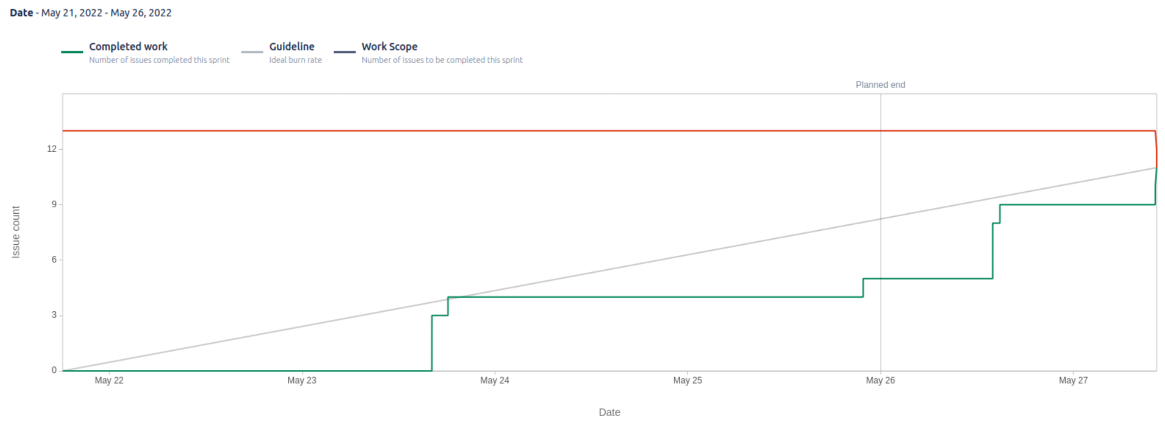
1. See previous order
    - She goes to her orders
    - On the list provided she clicks on the one she is curious about
2. See the status of the order
    - On the specific order tab she can see the details of that order

2.4 Project epics and priorities

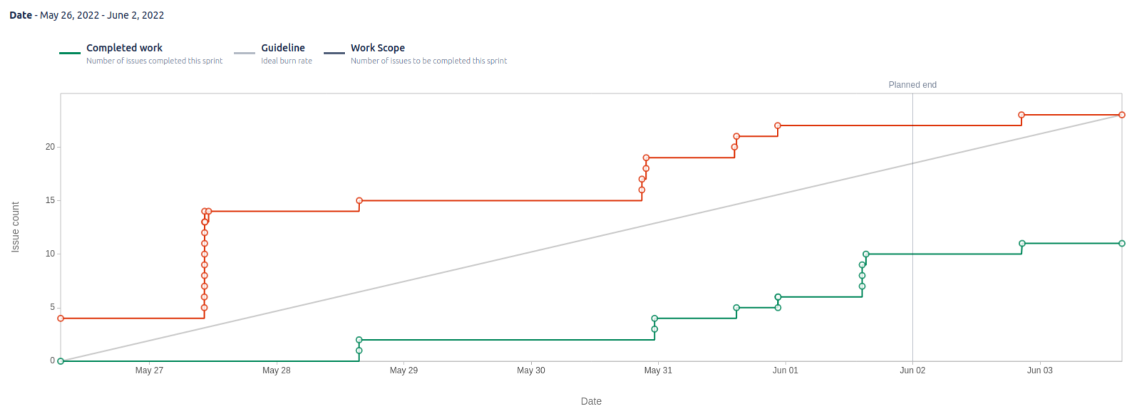
The project planning was organized with the goal of building the main support epics first, epics that are the core of the project and that have epics that are dependable on them. To build these support epics first the approach taken for the development lifecycle was focused on an Agile approach. This approach was made with 4 core sprints where in every sprint key epics were to be developed. An additional sprint 0 was made for the project concept to be decided and established.
The sprint zero burndown chart can be seen below:



The first sprint was focused on defining user stories, outlining SQE tools and practices, defining the product architecture, and building a UI prototype.
The sprint one burndown chart can be seen below:

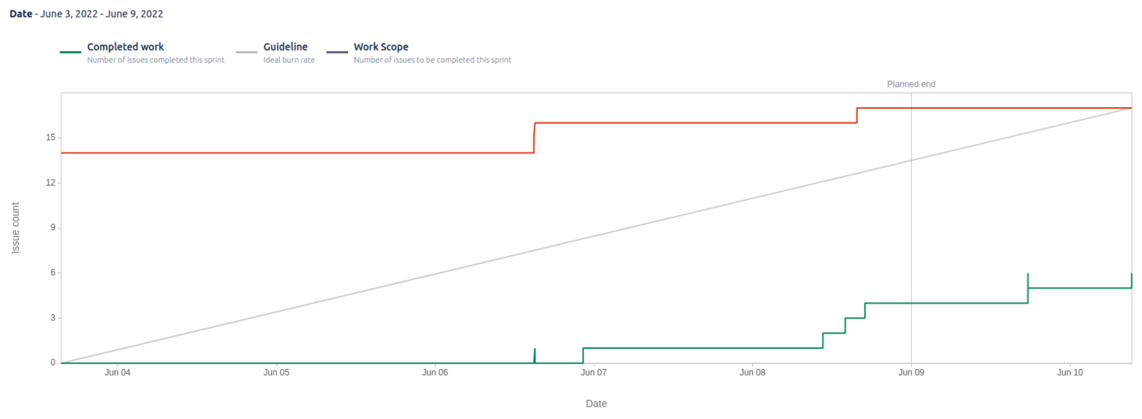


The second sprint was focused on the core epics. The first core Epics to be established and worked on were account registration, account management, and login management. With these key epics, other epics could be built upon. These epics also relied on infrastructure that wasn't built already, the development of this infrastructure and the CI pipeline was also built during this sprint.
The sprint two burndown chart can be seen below:



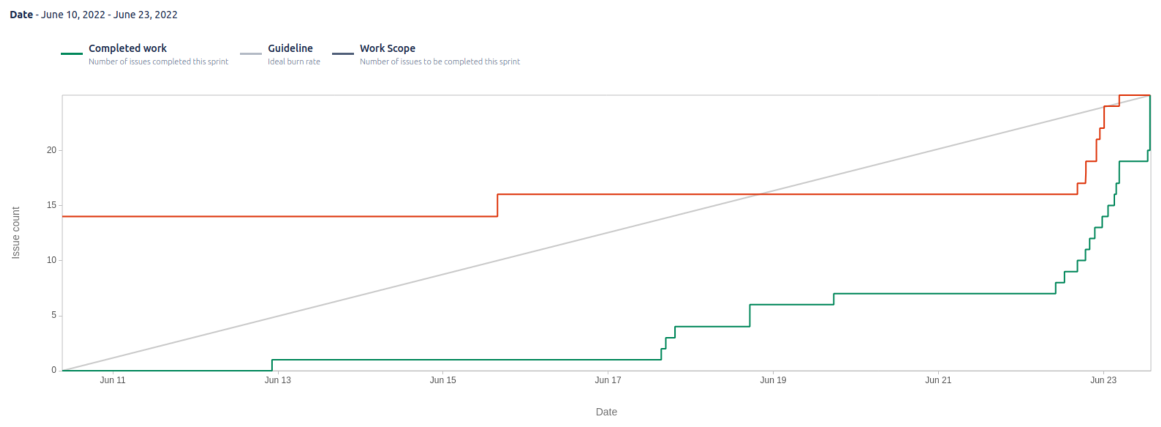
After the core epics and infrastructure took place, other epics that relied on the core ones could be developed. These epics were the complex delivery auction system, delivery creation, and management via UI or a light API. The CD pipeline and the final version of the QA Manual were also built during this sprint.

The sprint three burndown chart can be seen below:

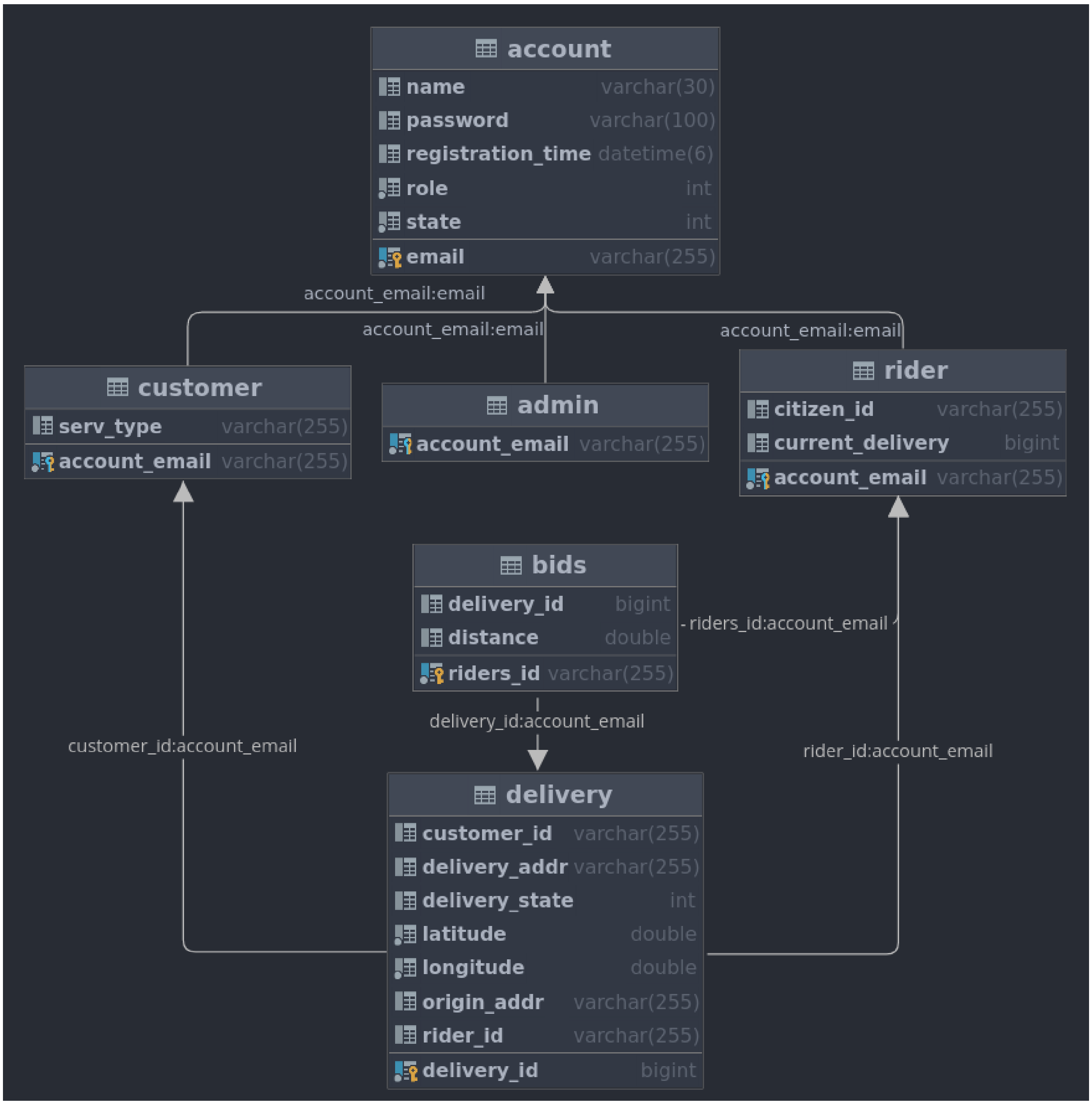


After the final core epics were made, other epics with less importance were developed, these ones being the development of the infrastructure for metrics gathering and visualization for the admin account, hardening of the API, development of a pub/sub connection between users and server for notifications and final UI optimizations and server deployments.

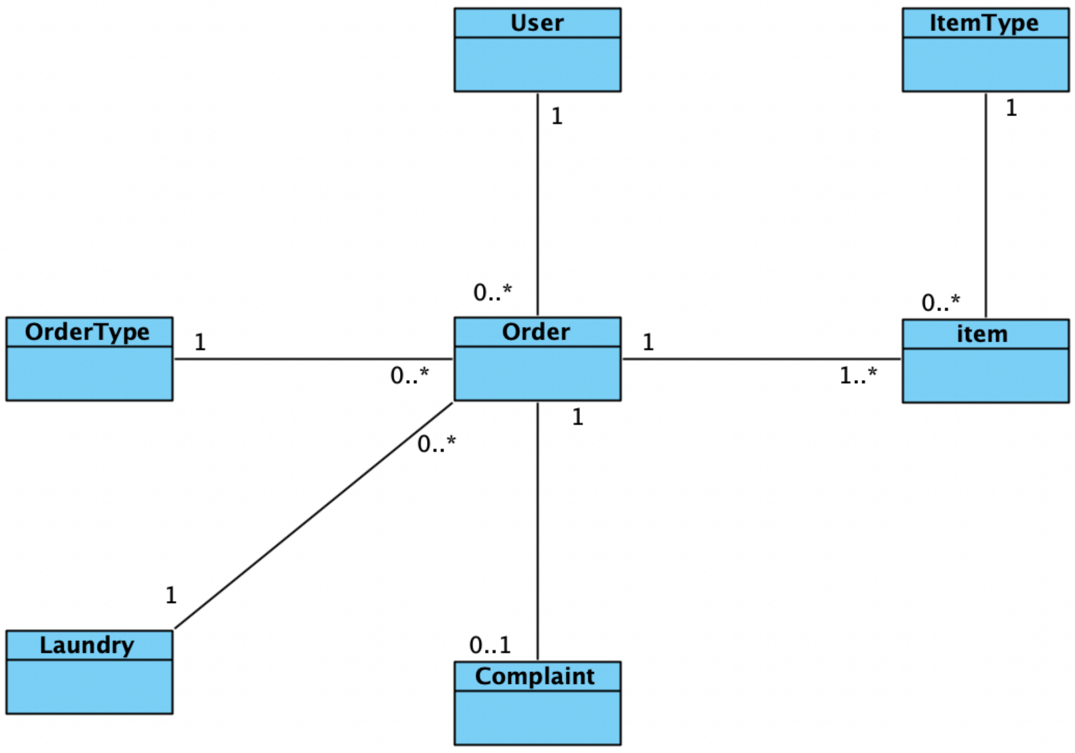
The sprint four burndown chart can be seen below:



3 Domain model



For the Qourier domain model, there are 3 different types of accounts (Administrator, Customer and Rider), all of which inherit attributes from the basic Account entity. The Delivery entity is associated with a Customer, and optionally a Rider if that Delivery ends up being assigned to one (it won't be assigned if the bidding process doesn't choose a Rider for that Delivery). Both the Rider and the Customer can have multiple Deliveries associated with them.
The Bids for the bidding system are also persisted, which are related to a specific Delivery and a Rider, with both having the ability to be associated with multiple Bids.



In the Laundry@Home domain model, there are 3 core entities, the User, the Order, and the Item. These relate to each other in different ways, the User has zero or more orders and an order has one or more items. Then we have two other classes, which define either the order type or the item type.
Finally, we have other auxiliaries that complete additional features of this particular app. Being Complaint that is associated with the order and the Laundry. In this case, we only have one.

4 Architecture notebook

4.1 Key requirements and constraints

Before we defined the system's architecture, we laid out the key requirements and constraints that we considered for the project:

- The Riders should be selected for a Delivery based on a dynamic bidding system, which selects the most appropriate Rider based on their location relative to the delivery's start location. Each bidding round takes a set amount of time
- The Rider delivery acceptance notifications should be done asynchronously so that they are not dependent on doing refreshes or constant API calls
- The delivery status updates should be quick and up-to-date, delivered as notifications without the need for querying the delivery engine
- Both systems should have a correspondent mobile application: for the Qourier delivery engine, Riders should be able to apply their operations on a mobile app, and for the LaundryAtHome service, the users should be able to make orders and file complaints. Both apps should also contain an Admin environment
- The Qourier and LaundryAtHome mobile apps should be integrated with a service to obtain the user's location

4.2 Architectural view

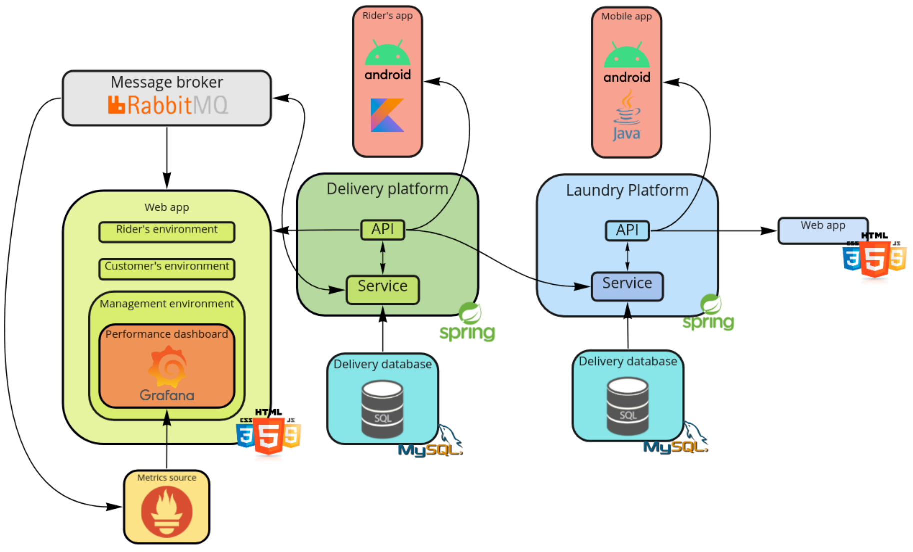
In the LaundryAtHome API we have 4 different requests mappings:

- AuthController
  - Handles the login and registration process, and is here that we set a JWT cookie to keep track of the user session
- Main Controller
  - Mainly, is here where we get the HTML pages, sometimes with a required object in order to fulfill the information required to present the page to the user
  - In this controller we have the tracking endpoint as well, this one before returning the page, it made a request to Deliveries Platform in order to get the state of order.
    It also gets the information from the RabbitMQ as well but due to technical problems wasn't working and it is provided in comments on the code.
- Order Controller
  - In this controller processes all the operations required to perform an order, either by making an order, a complaint, or canceling.
- Mobile Controller
  - In this controller all the operations described above are performed but with the necessary changes to fulfill the needs of the mobile app.

In the service package, it is where it is provided the internal logic operation on the data and its respective access to the repository. All the controller except the authentication one uses the OrderService.

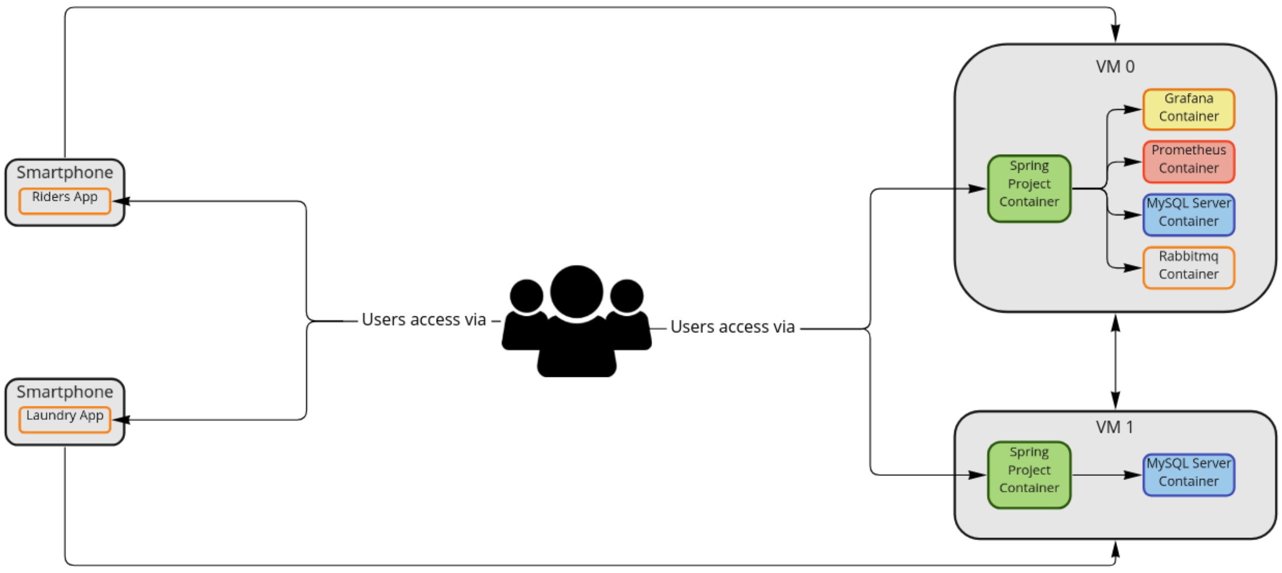
The Qourier package is where it is performed the logical operation between the Qourier App and the LaundryAtHome.

On the Qourier project, it is used two main controllers. The API Controller is responsible for the API calls handling, and the Web Controller, being this one responsible for the Web interface provided by the project.
The project also relies on Prometheus and Grafana to provide visualization of metrics and the status of the hosting system, these components also allow getting metrics for the status of the RabbitMq component. The RabbitMq component allows asynchronous notifications and pub/sub communication between the project and the clients.
Both projects have a database component that enables persistence, these components are a MySQL database that each project independently has.
An architecture diagram can be seen below:

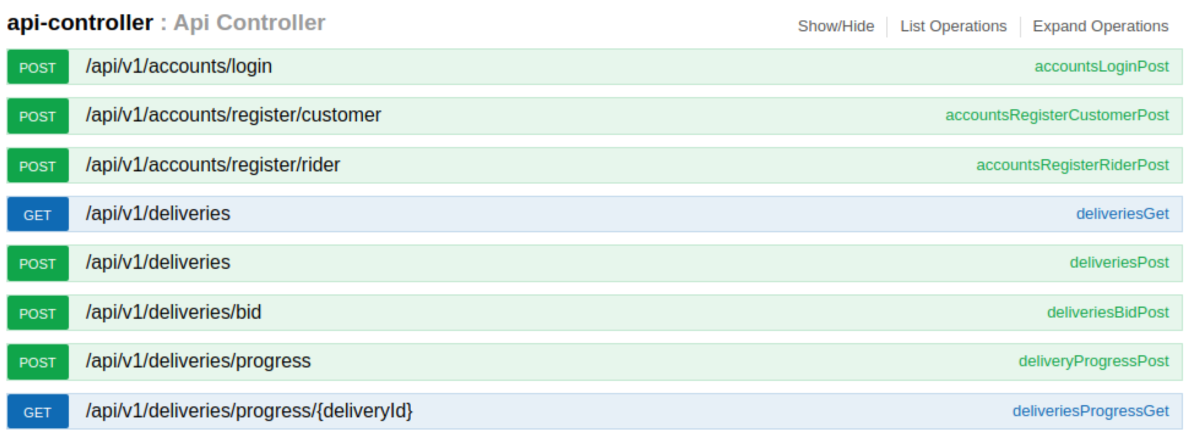


4.3 Deployment architecture

The deployment of the main project and the subproject was made with the help of Github actions, enabling on Pull Request to deploy a new version of the projects to the host machine. This CD pipeline allows faster deployment of new versions.
The host machines used for deployment were provided by Azure, it is used two virtual machines (VMs) to host the project and subproject, and the communication between them is made with the help of a REST API.
The complementary Android applications can be installed with the apk provided by the project and communication with their main projects is made with also the help of a REST API.
The Qourier and the LaundryAtHome rely on Docker containers for easy deployment, being both projects being hosted with a Spring container and a Mysql DB container. The Qourier project also has complementary dockers to better retrieve metrics of the system and to enable pub/sub communication with the clients.
The deployment architecture can be seen below:



5 API for developers



POST	/api/v1/accounts/login	accountsLoginPost
POST	/api/v1/accounts/register/customer	accountsRegisterCustomerPost
POST	/api/v1/accounts/register/rider	accountsRegisterRiderPost
GET	/api/v1/deliveries	deliveriesGet
POST	/api/v1/deliveries	deliveriesPost
POST	/api/v1/deliveries/bid	deliveriesBidPost
POST	/api/v1/deliveries/progress	deliveryProgressPost
GET	/api/v1/deliveries/progress/{deliveryId}	deliveriesProgressGet

METHOD	Endpoint	Description
POST	/api/v1/accounts/login	Used to make a login request and retrieve an API token.
POST	/api/v1/accounts/register/customer	Allows the registration request of a new customer account
POST	/api/v1/accounts/register/rider	Allows the registration request of a new rider account
GET	/api/v1/deliveries	Used to retrieve the deliveries present on the system
POST	/api/v1/deliveries	Used for the creation of new delivery request with the help of an API token
POST	/api/v1/deliveries/bid	Used for the registration of a bid from a rider to deliver a delivery
POST	/api/v1/deliveries/progress/	Used to update the progress of a given delivery being delivered
GET	/api/v1/deliveries/progress/{deliveryid}	Used to retrieve data on the state of a given delivery, being the delivery if from the delivery of interest given as an argument

6 References and resources

Testing of asynchronous and timing-dependent behavior: http://www.awaitility.org/
System metrics tracking: https://www.rabbitmq.com/prometheus.html
Deliveries platform API requests: RestTemplate (Spring Framework 5.3.21 API)