Qourier & LaundryAtHome

Martinho Tavares, 98262
Tomás Candeias, 89123
Afonso Boto, 89285
Rodrigo Lima, 98475

23.06.2022

Part 1: implemented products and services

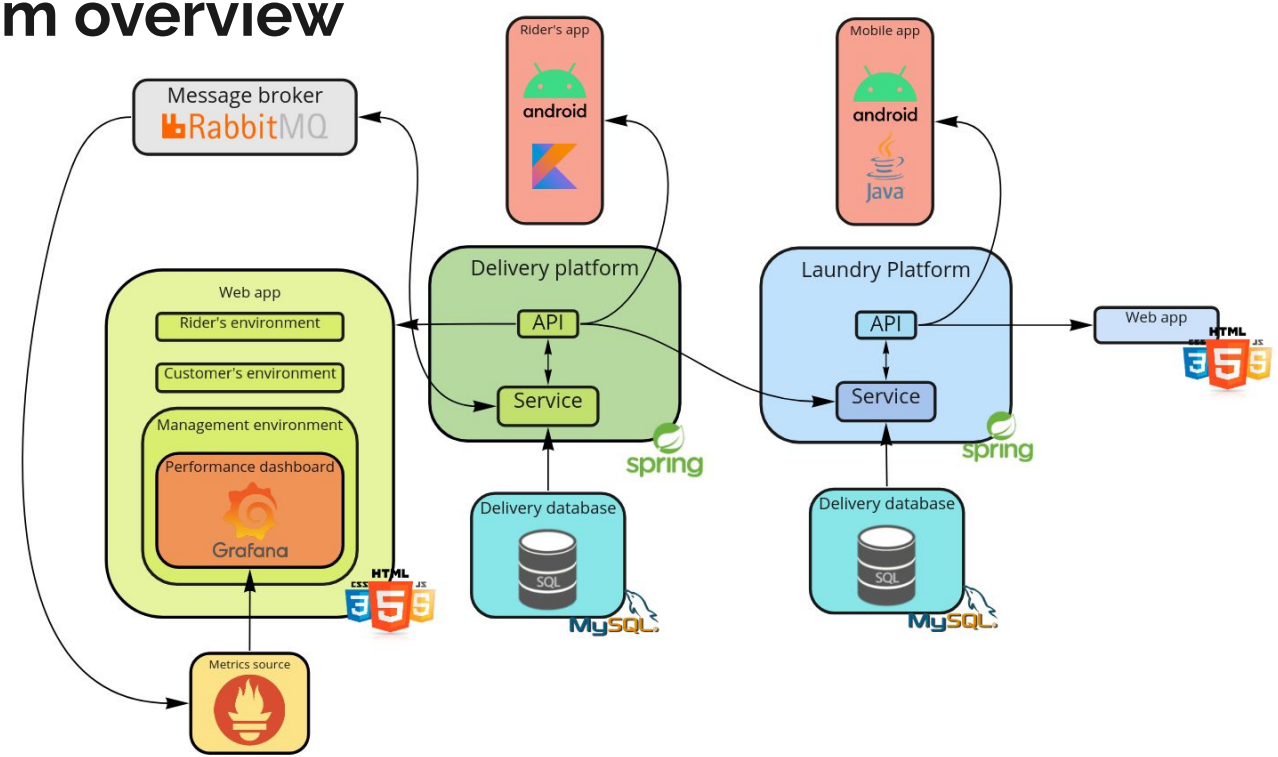
Qourier platform: key features

- Web app for management by Riders and Customers (delivery businesses)
- Approval-based registration system managed by the administrators
- Account lifecycle that supports suspension of accounts, managed by the administrators
- Dynamic bidding system for delivery assignment to Riders based on distance to delivery origin
- Asynchronous messaging for notifying Riders if they were assigned to a job
- Tracking of delivery progress by Administrators and Customers on a publish-subscribe fashion
- Admin page for monitoring platform status

PoC integration: LaundryAtHome business

- Webapp with a simple and intuitive login/register system for the customers
- Choose one of four kinds of services (Wash, Dry Cleaning, Wash & Laundry, Special Items Laundry)
- Make an order with an address and a list of items, the user may choose as many as he wants
- A customer can see all the orders actives and completed
- Tracking of a specific order
- See details of a delivered order (completed)

System overview



System flows and B2B integration

- When the customer submits an order
  - A POST Request is made for the endpoint /api/v1/deliveries with the request parameter basicAuth with the token of the login of our Laundry
  - The response for this request should be a json object with the entire delivery object
- When a user sees the details of a specific order
  - A GET Request is made for the endpoint /api/v1/deliveries/progress/ with the path variable corresponding to the deliveryId and the state of the order is the response of this request
  - If the state is “DELIVERED” the order is completed and the user can not add a complaint and now can see the delivery date
  - Note: RabbitMQ implementation and service logic is currently in comments due to a connection problem

Products demonstration

Current 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

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

Part 2: development workflow

How did we work?

Agile project management and work assignment

Story-driven development, along 4 iterations, managed in Jira.

Cooperative development and integration of new increments

Gitflow workflow was followed. Pull requests followed a naming scheme, and should be peer-reviewed before merging.
New increments (pull requests) trigger automated tests and static code analysis, and these should pass before they are accepted.
The static code analysis tool used was SonarCloud, and the quality gate used is the SonarWay, which focuses on newly developed code.

CI/CD strategy

GitHub Actions was used to implement the pipelines.
The following workflows were defined for CI:
- **Auto-release**: create a **release** branch every Tuesday (2 days before the end of a sprint), and merge both the development branches (**qourier_dev** and **laundryathome_dev**) into it. After the merges are done, a Pull Request is created for merging the **release** branch into the **main**. The Product Owner is automatically assigned as a reviewer for this Pull Request
- **LaundryAtHome** and **Qourier**: on every pull request to the **laundryathome_dev**/**qourier_dev** branch, apply the Google Java formatting, run all the tests and do static code analysis. One of the developers responsible for this part of the project, except the one that authored the pull request, is automatically assigned as a reviewer to the created pull request
- **Main-check**: on every pull request to the main branch, apply the Google Java formatting, run all the tests for both projects and do static code analysis

For CD, a single workflow is run when the **Main-check** workflow is completed, which will deploy the projects on an Azure VM.

Highlights on the implemented SQA strategy

Strong points:

- Automation on all tests developed for the respective part of the project

- Static code analysis that enforces a quality gate on all increments

- Once the project has minimum functionality, parallel development of new features is easy

- Forced BDD approach allows for a greater focus on application functionality rather than internals, which permits quick and effective testing of the system behavior

- Organized process allows easier tracking of previously developed features and mapping them to User Stories

Weak points:

- Coding style ended up having to be fixed automatically instead of being enforced, which can lead to merging conflicts if developers forget to pull the automated changes

- The upfront application of BDD takes focus away from the development of Unit and Integration Tests, as static code analysis tools and test reports will present good coverage and passing tests, even though the application's internal modules may present quality issues

- No distinction between Unit, Integration and Functional tests in the automated QA processes

- It's hard to kickstart development, because very few features have been developed. This made development awkward as few branches could be done in parallel since they all depended on functionality that still had to be implemented

Extra credit challenges