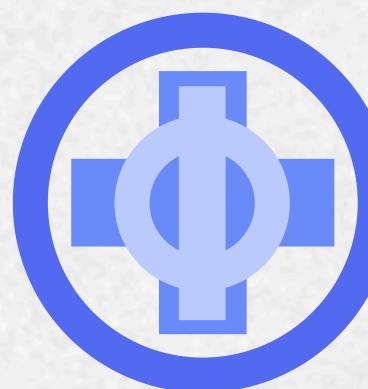


PHIHUB - MEDICAL MANAGEMENT APP

TQS GROUP PROJECT



PHIHUB

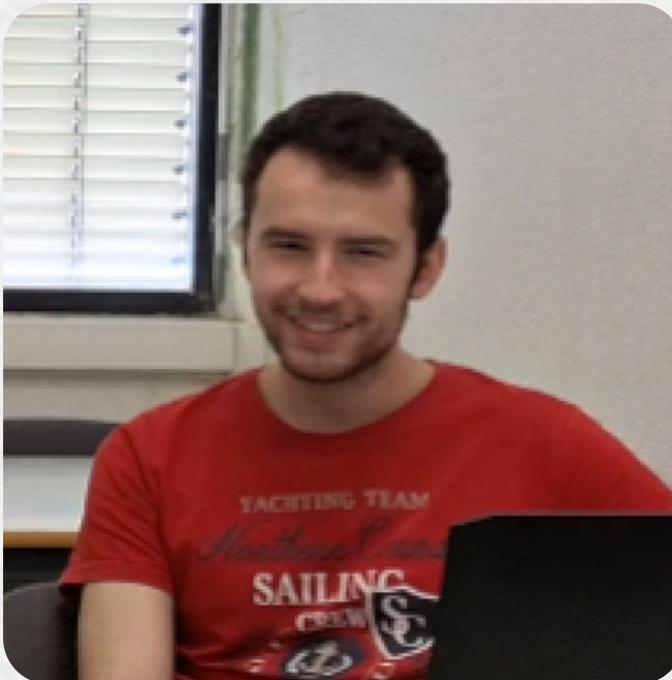
Daniel Madureira [107603]
João Luís [107403]
Pedro Ramos [107348]
Rodrigo Aguiar [108969]



OUR TEAM



Daniel
Madureira
QA Engineer



Rodrigo
Aguiar
Team Manager



Nuno
Luís
Product Owner

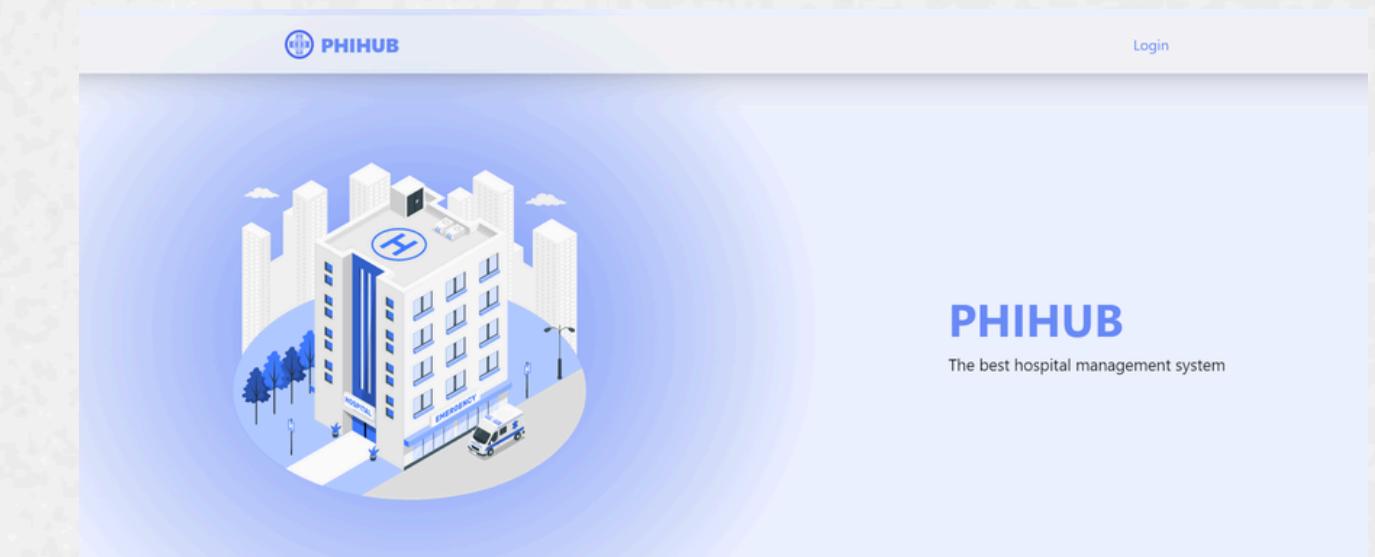


Pedro
Ramos
DevOps Master

PRODUCT CONCEPT

An integrated ICT solution to handle:

- Patient's appointments;
- Appointment's management;
- Staff management;
- Medic's creation;
- Queue lines and priority lines.

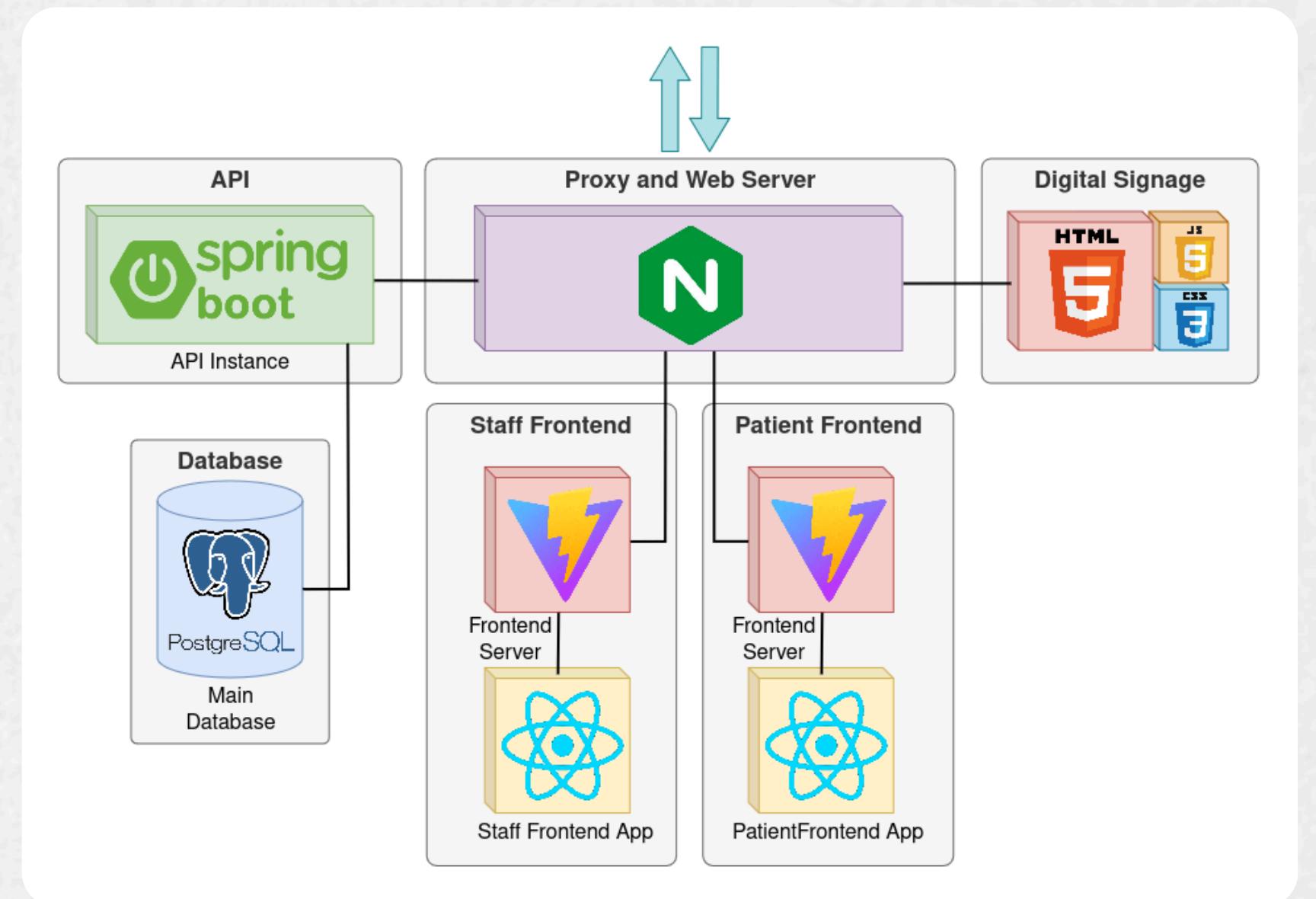


The screenshot displays a mobile application interface for 'PhiHub'. At the top, there is a header bar with the text '@PhiHub' on the left, 'Home' and 'About' links in the center, and a 'Login' button on the right. Below the header, there is a large, semi-transparent background image of a hospital building. On the right side of the screen, there are several dark blue cards with white text and icons. One card is titled 'PhiHub Patient Office' with subtext 'Here you can mark appointments or search for your old appointments.' It has two buttons: 'Mark a new Appointment' and 'Check my old appointments'. Another card is titled 'Now Serving' and shows 'Desk: -'. A third card is titled 'Next Number' and shows '---'. To the right of these cards, there is a section titled 'PhiHub - 23:04' with a sub-section 'Previous Calls'. This section contains four smaller cards, each with a 'Desk:' label and a red circle with a minus sign inside, indicating no current calls. The overall design is clean and modern, using a color palette of blues, whites, and greys.

ARCHITECTURE

The final system consists of:

- A NGINX Proxy and Web Server;
- A general Backend;
- A Patient oriented Frontend for clients;
- A Staff oriented Frontend for medics and staff personnel;
- A Digital Signage for waiting room screens.



TOOLS



JIRA

Issue Tracker

Integrated with Xray to import automatic test results to JIRA



GITHUB

Remote Code Repository

Various components organized into submodules with a main repository, which includes all submodules.



Project Management

Agile Development with Weekly Sprints

Define what user stories we will work on

Create subtasks for each user story in JIRA

For each subtask create a Feature Branch

Develop the wanted code and tests

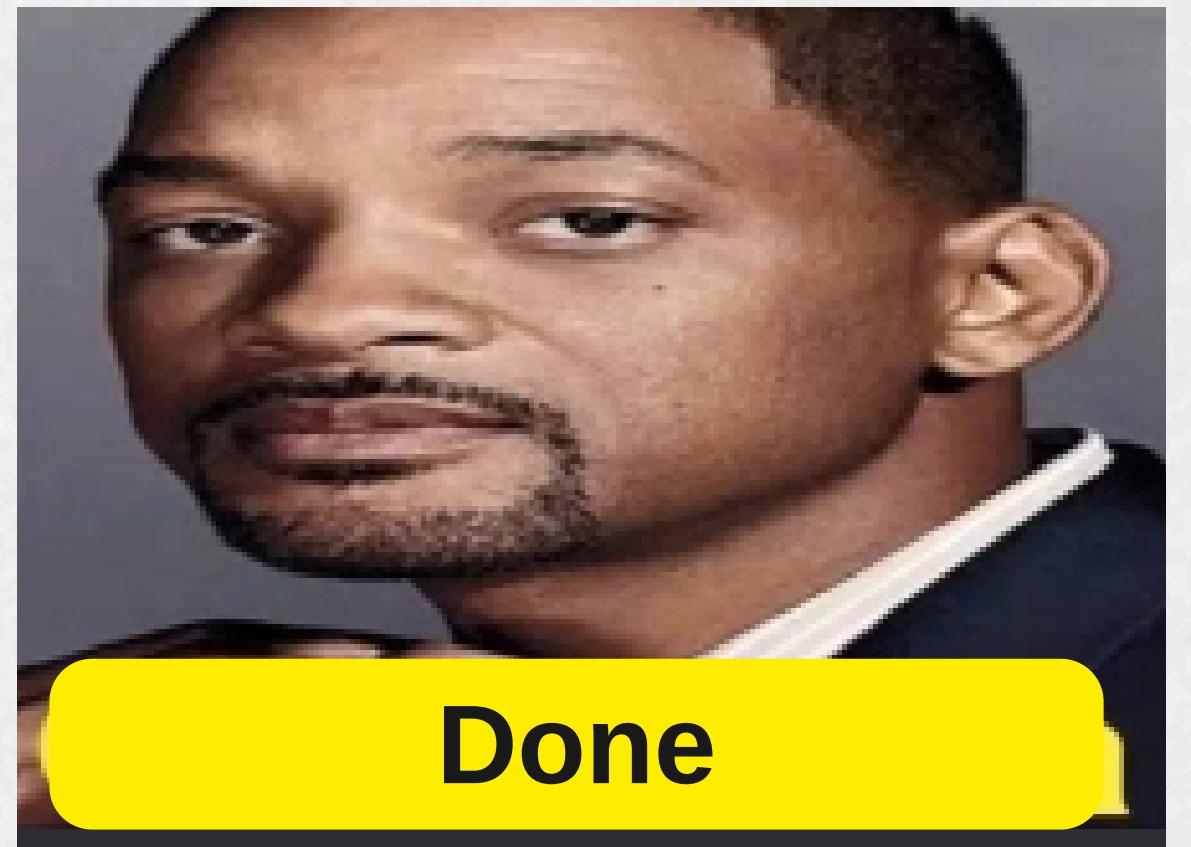
Create a pull request to the main branch of the submodule

The screenshot shows the Jira interface for the 'PhiHub' project. The left sidebar has sections for PLANNING (Timeline, Backlog, Board, Calendar, List, Reports, Goals), DEVELOPMENT (Code, Security, Releases), and OPERATIONS (Deployments). The main area is titled 'Backlog' and shows a list of user stories for 'PHIH Sprint 4' (24 May – 31 May). There are 13 issues listed, all marked as 'DONE'. The issues include tasks like 'Pay Appointment Bill - Patient', 'Issue Appointment Bills - Staff', 'Develop presentation diagrams', and 'End appointment - Medic'. Each issue has a small icon, a progress bar, and a 'Done' button.

Definition of Done

To define if a user story is done we agree to the following rules:

- Developed solution meets all user story requirements
- Supplied with automatic unit, integration and functional tests
- Test coverage of at least 80%
- Code reviewed by at least 1 other team member
- No bugs or security issues

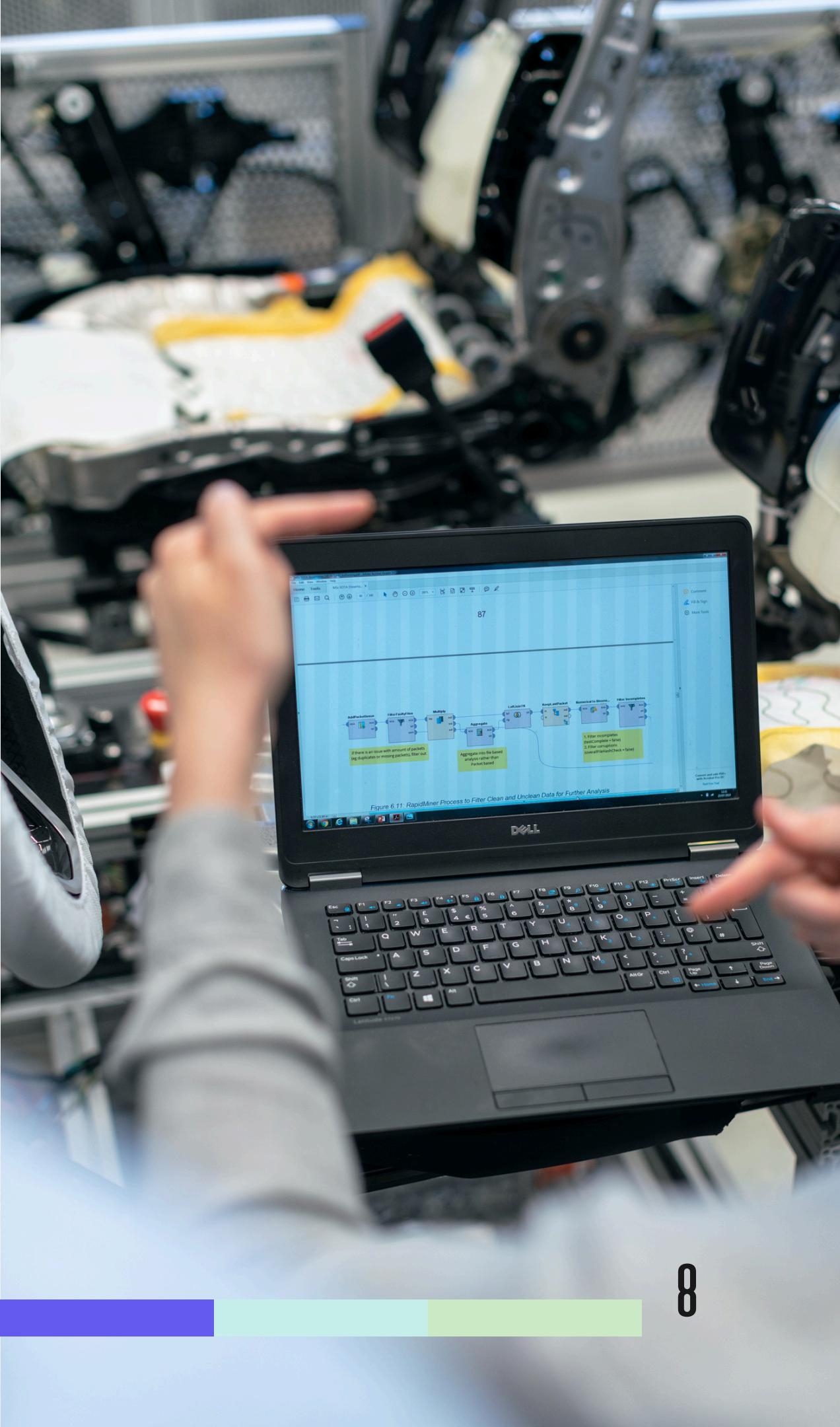


Testing Plan

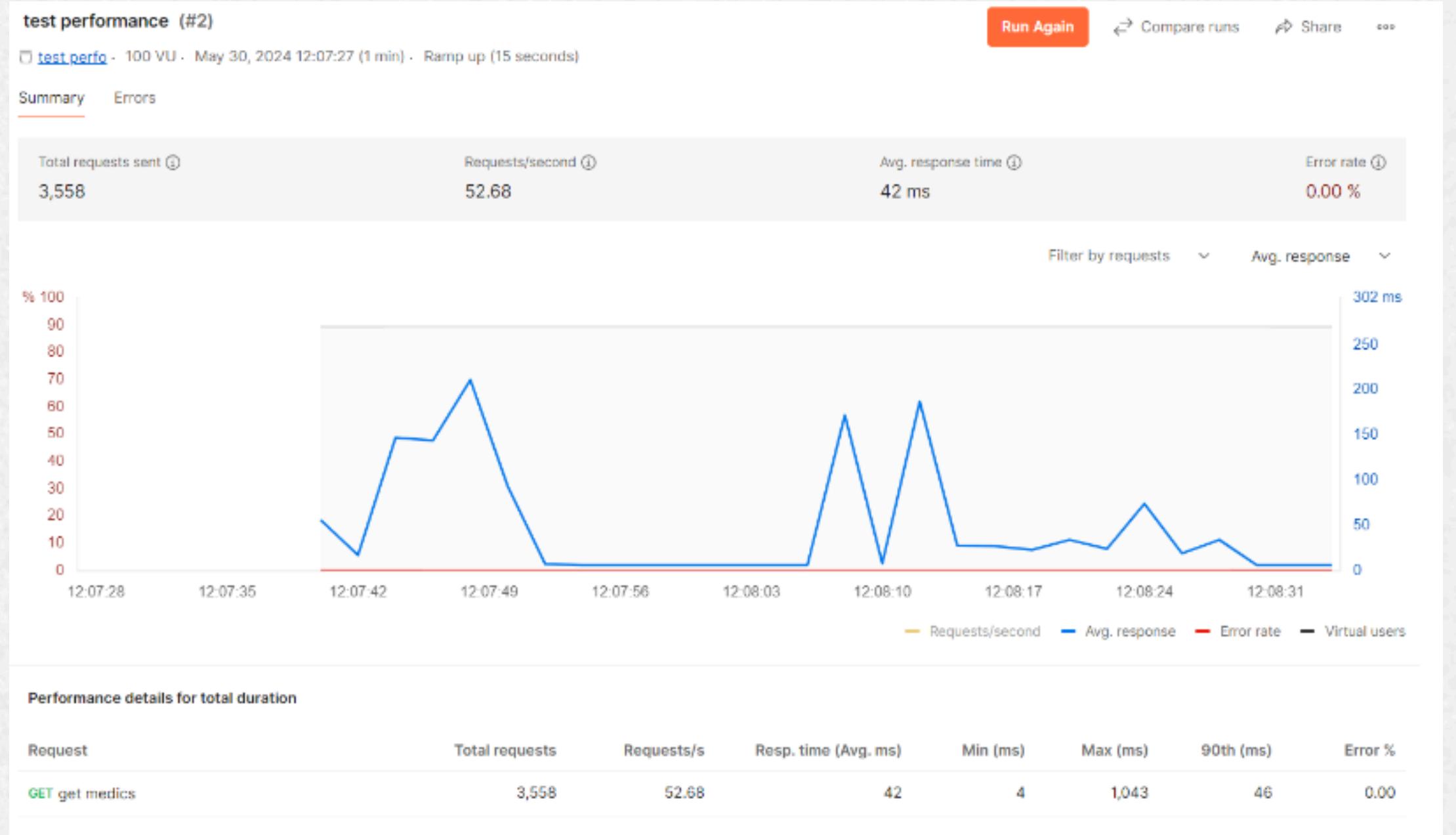
For general testing, unit tests were developed to cover:

- The repositories;
- The services;
- The controllers;
- The full integrated backend;

Additionally, frontend and full application tests were created to allow testing the most common user workflows, as to ensure the integration of the systems and the frontend application work properly.



Performance Testing



Simulated load with 100 virtual users

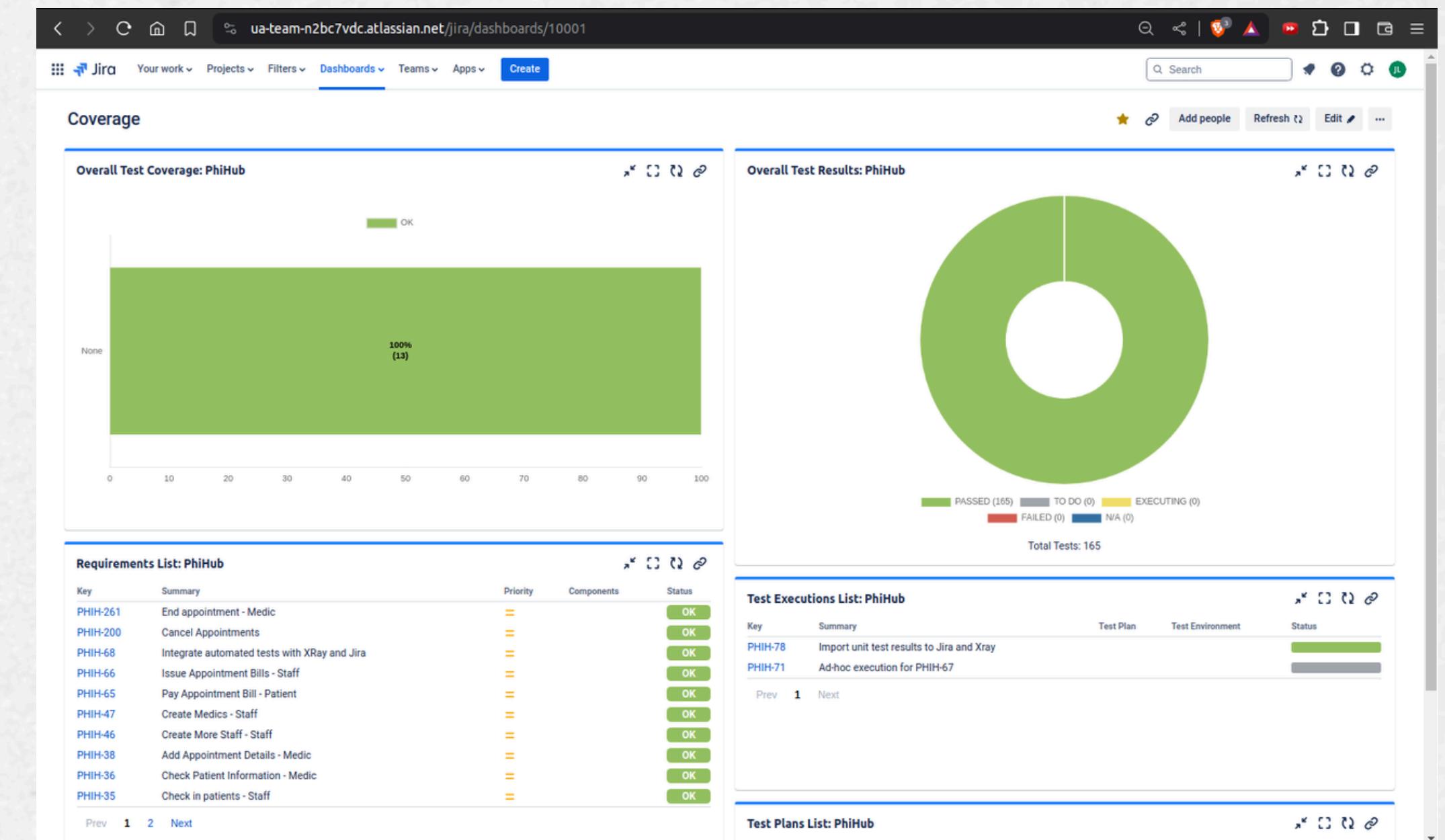
Total of 3558 requests in 1 minute

Average of 52 requests per second

Average response time: 42 ms

Quality Assurance

X-ray - Native Test Management for Jira;
Grouped test in test sets;
Linked various test sets to Issues;
All issues covered and all tests passed;



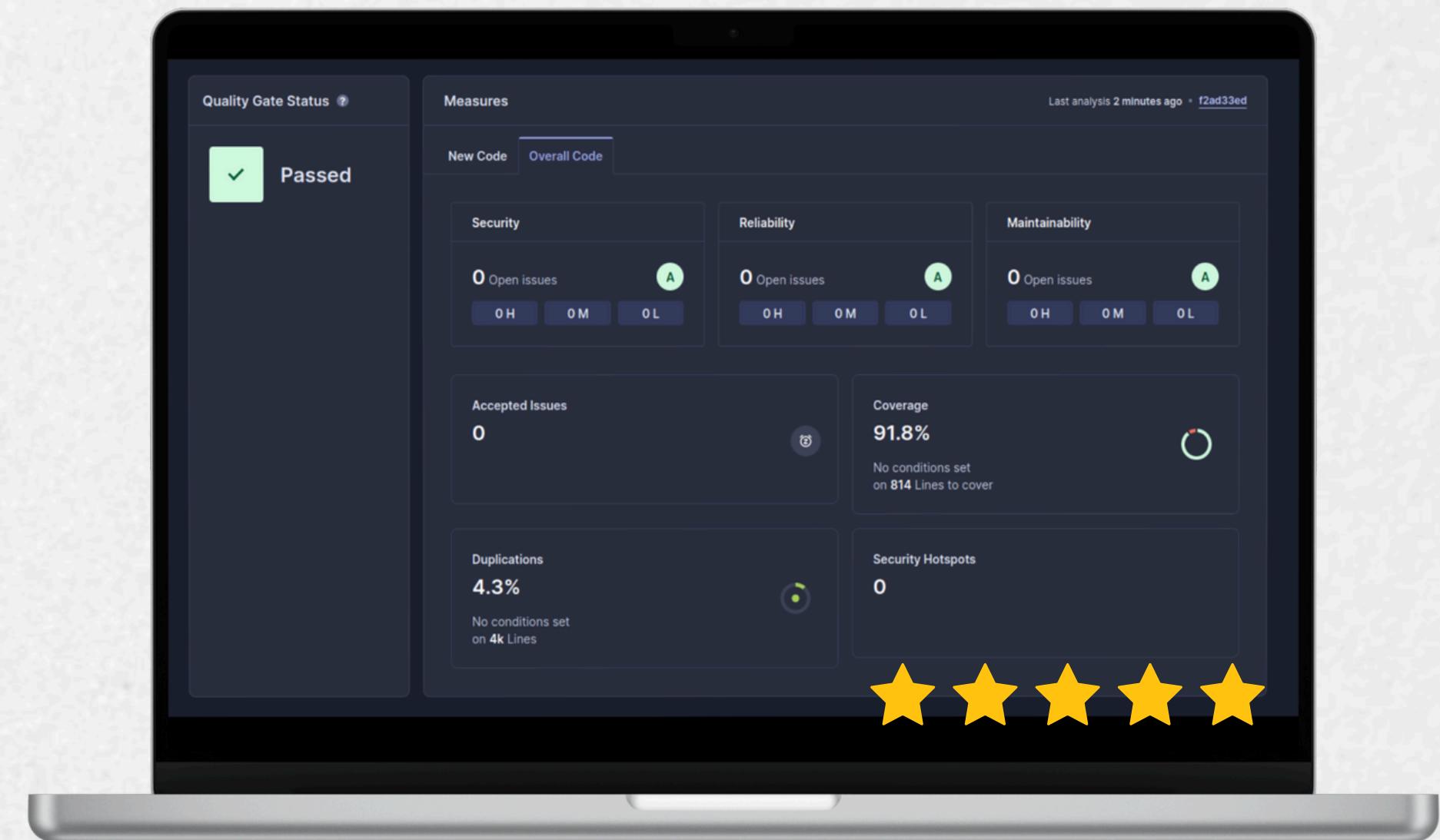
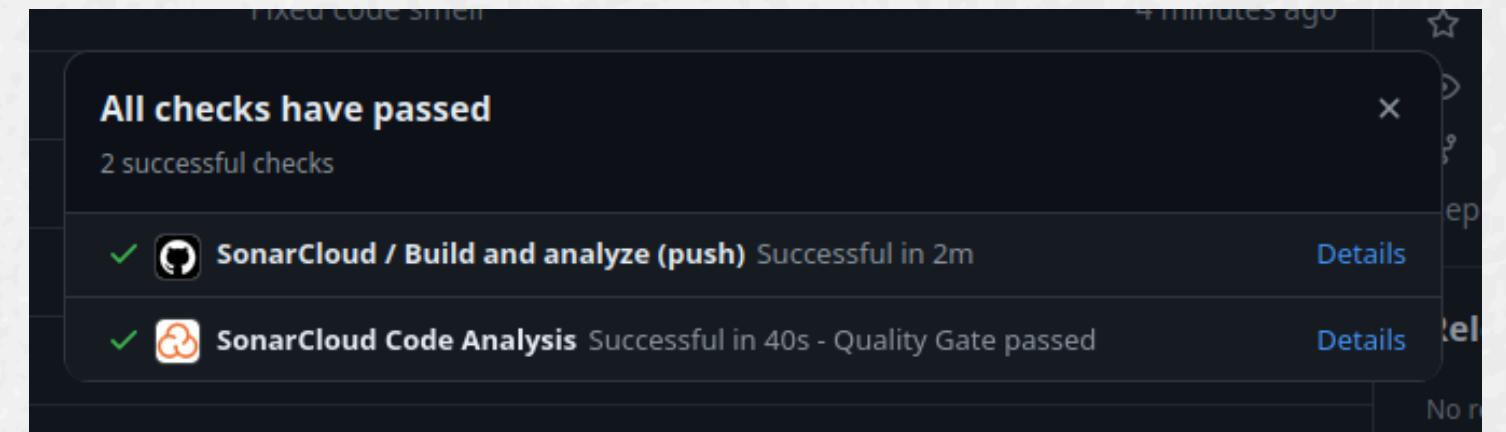
CI / CD

AUTOMATIC TESTING SONARCLOUD ANALYSIS

On every push or merge, the implemented tests are executed automatically to ensure that no broken or bad changes go to production.

The results are then published to SonarCloud, a static code and test result analyzer that makes it easy to check for problematic code changes.

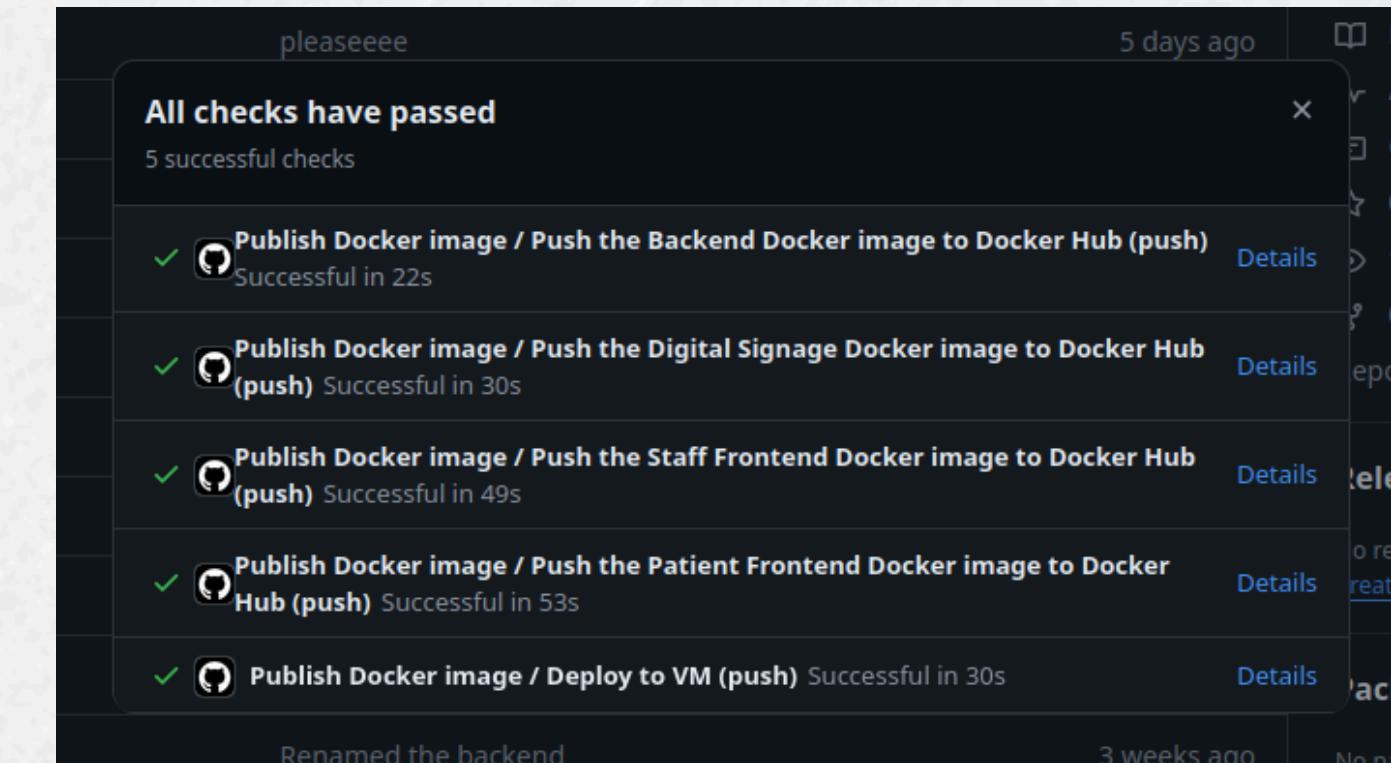
Sonarcloud code analysis is done in both frontend and backend components.



CI / CD

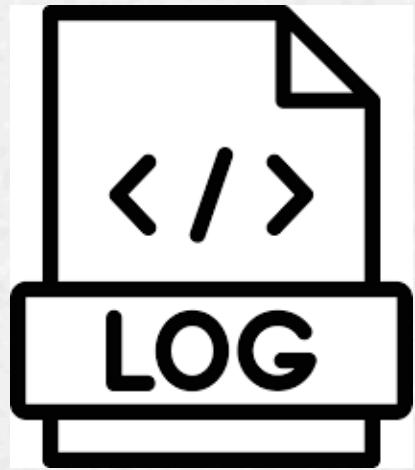
AUTOMATIC DEPLOYMENT

- Build and push Docker Images to DockerHub
- Self Hosted Github Runner running on the Virtual Machine
- Pull the latest images from DockerHub and build them



```
deploy:  
  runs-on: self-hosted  
  name: "Deploy to VM"  
  needs: [build-backend-image, build-signage-image, build-staff-image, build-patient-image]  
  
steps:  
  - name: "Checkout"  
    uses: actions/checkout@v4  
  - name: "Deploy"  
    run:  
      echo "CREATE DATABASE phihub;" > init.sql  
      docker compose -f docker-compose.prod.yml pull  
      docker compose -f docker-compose.prod.yml up -d --remove-orphans  
      docker container prune -f  
      docker image prune -af  
      docker builder prune -af
```

System Observability



Logging

Relevant action logging with Logback

Logging to both console and file



Healthcheck

API Healthcheck with Spring Actuator

/actuator/health endpoint



Log analysis

Various attempts were made at integrating Spring with an ELK Stack

Unfortunately, they were unsuccessful =/



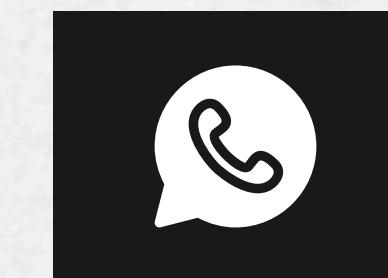
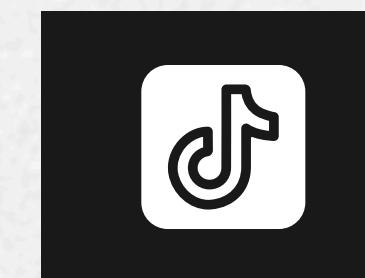
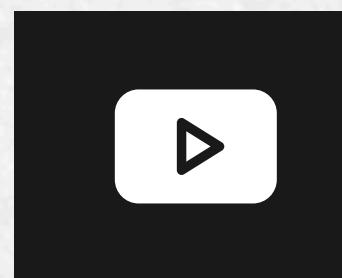


DEMO

THANKS!

Do you have any questions?

deti-tqs-18.ua.pt



CREDITS: This presentation template was created by [Slidesgo](#),
including icons by [Flaticon](#) and infographics & images by [Freepik](#)