Project Plan

revision A, November 4th, 2022

Project Code Defenders - Robo Tournament

Team Codebenders

Agenda

- Our Team
- Project vision and plan
  - About Code Defenders
  - Project requirements
  - Our solution
  - Motivation
  - Customer
  - What the project is not going to address
  - Risks
- Our way of working
  - Implementation of SCRUM
  - Tools for communication, project management and development
  - Development and testing workflow

Our Team

Frontend

Product Owner

Fanny Delnondedieu

Fabio Patella

Scrum Master

Dominik Brdar

Testers

Hrvoje Rom

Backend

Simone Mezzaro

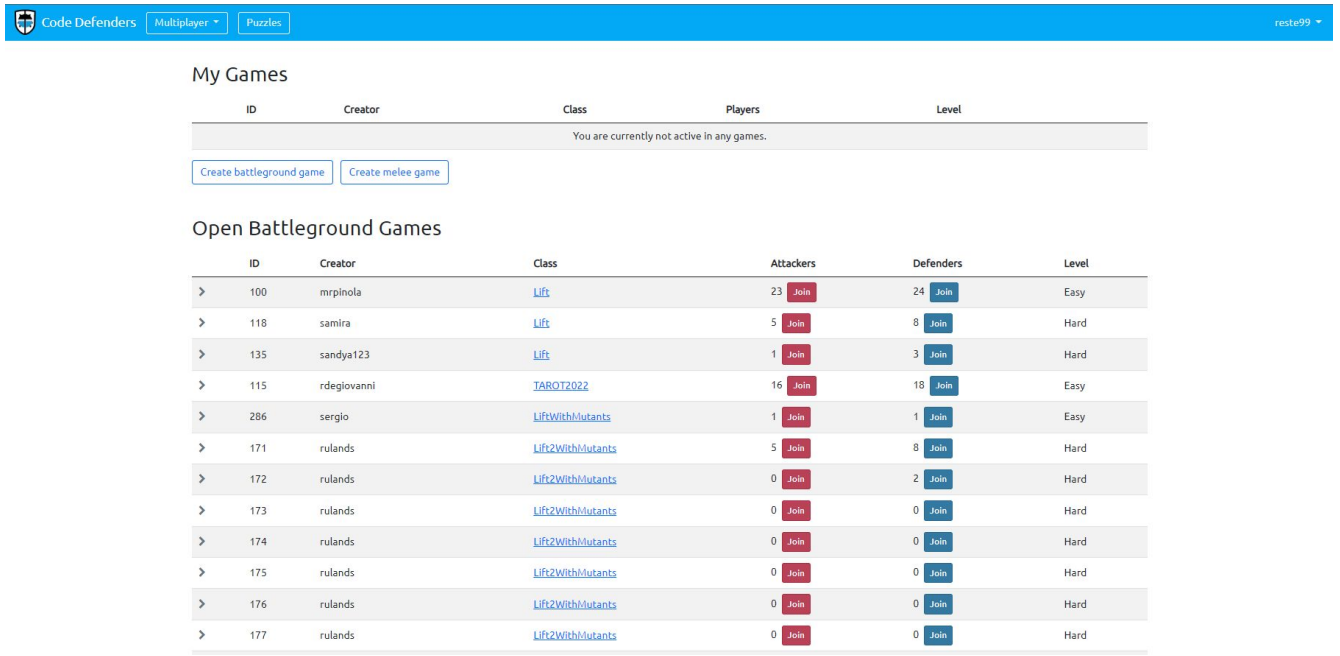
Riccardo Nava

Andrea Restelli

Project vision

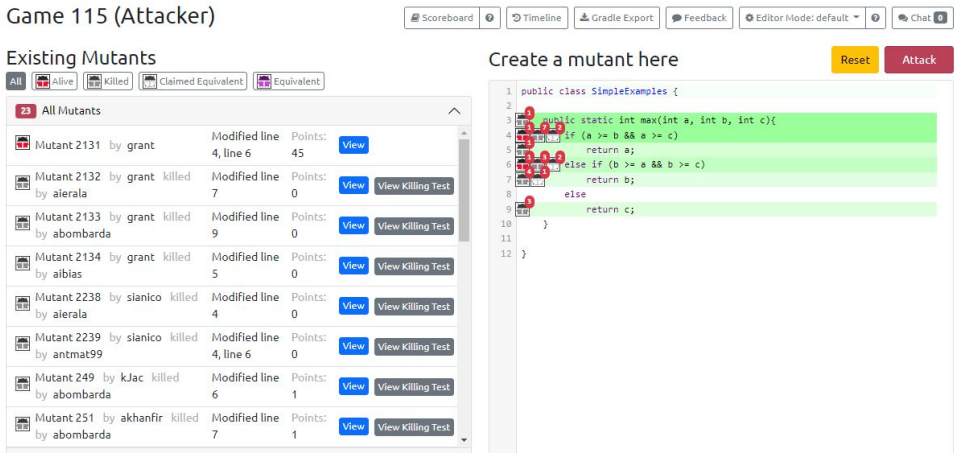
Code Defenders

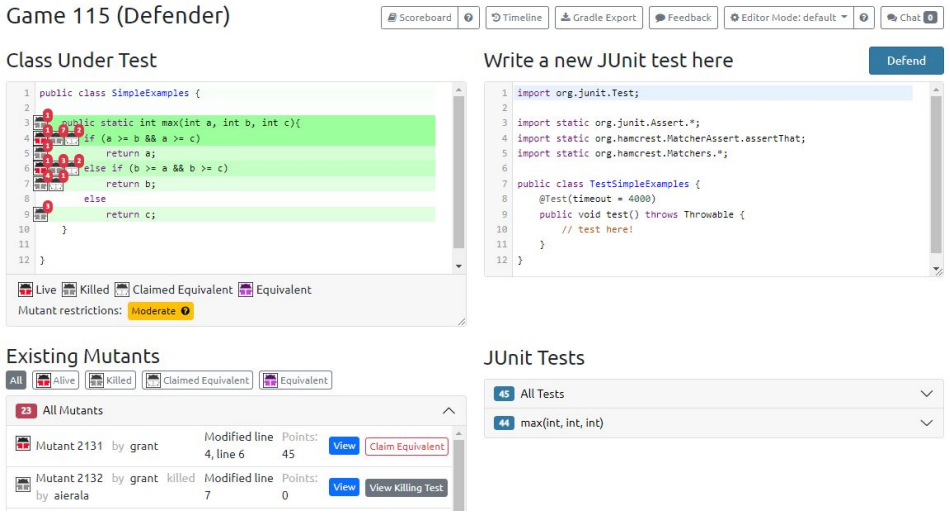
The **“CodeDefenders: RoboTournament”** project aims at enriching the already existing game of CodeDefenders with some new functionalities: students tournaments, games against bots, and live score streaming. The tournaments function will be implemented by an **external application**.



About Code Defenders

CodeDefenders is a **web game** on mutation **testing**. Each game involves two teams of students, competing against each other, and a piece of code (code under testing).





The attackers team injects faults in the code under testing creating so called mutants, whereas defenders write unit tests to spot the injected problems.

Defenders earns points for each discovered mutant. Attackers earn points for each mutant escaping the tests.

Project requirements

- Design and implement a set of **OpenAPIs for CodeDefenders** which can be used from the tournament application to manage games and players.
- Implement the **tournament application**. This application must use CodeDefenders as a remote service (through the APIs) and must include at least two tournaments modalities.
- Implement a **streaming** component which allows users to follow in progress games live. This component can optionally include an “overall tournament view” showing schedule, standings and other information for each tournament.
- Design and implement a set of **APIs** which allows users to train **bots** over past games data and to let those bots play CodeDefenders.

What is our solution?

Since CodeDefenders is an open source project deployed on a public GitHub repository, we firstly plan to create a **forked repository** where we can inspect the code, identify which functionalities need to be exposed and implement APIs.

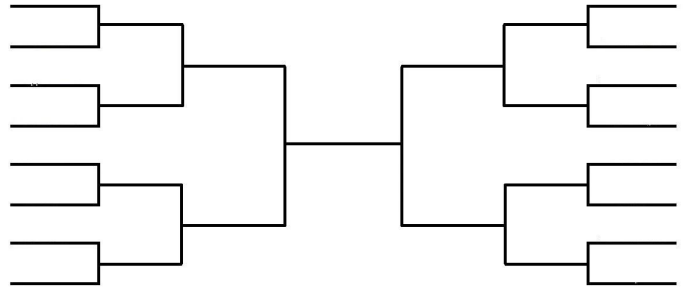
CodeDefenders backend is implemented using Java Servlet technology and our APIs will conform to it.

Our tournament application will be a **web based application**. The frameworks and the technologies employed to realize it will be discussed in more details with the customer during the next meeting. We plan to include the streaming component in the tournament application itself.

Motivation

Software quality and testing are at the heart of software engineering, but they are not always at the heart of software engineering education. CodeDefenders proposes the use of **gamification** to teach **mutation testing** and to strengthen testing skills, by introducing a mutation testing game. CodeDefenders can assist educators in delivering complex mutation testing concepts and is intended to make the learning experience more **enjoyable** and **fruitful** for students.

To make the game even more fun, the scope of the project is to introduce a **tournament mode** in it, so that players can compete and to climb the leaderboard.



Because tournaments are not fun without an audience, we are asked to implement also a **streaming service/application** that let guests and users follow the games of their preferred players live.

Lastly, the design of an API to allow the integration of **bots** in CodeDefenders would make possible to create challenges where students/players/teams compete against artificial intelligence.

Our customer

Professor **Alessio Gambi**
IMC University of Applied Sciences Krems (Austria)
University of Passau (Germany)

Worked on CodeDefenders in the past, now he has not enough time to implement anything.
The project is currently maintained by some students from University of Passau.
However, he will be our **reference point** for the RoboTournament extension.

We will join the students' Slack workspace to ask more technical questions **directly** to the current maintainers of the project.

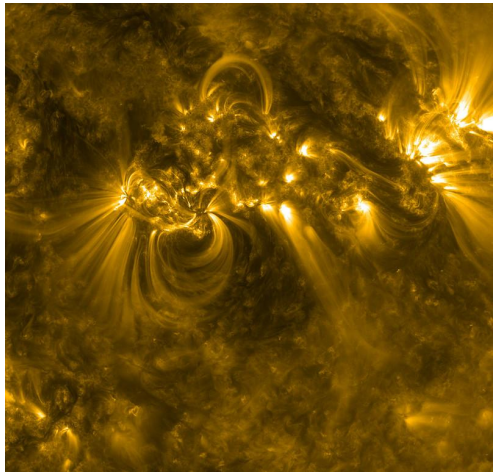
First meeting with the customer on Wednesday. Two other students will join the meeting together with the Product Owner in order to avoid misinterpretations.

What the project is not going to address

- The tournament application will be an **external application**, developed separately. It won't be a plugin of CodeDefenders nor an application running on the same host.
- The tournament application will implement only the **tournament** and **streaming logic**. It won't reimplement or modify in any way the game logic, which is already coded in CodeDefenders and will be accessible through our APIs.
- We won't implement an **AI** playing CodeDefenders. This project requirement is **optional** and we are not planning to realize it because of the current lack of AI knowledge within our team.

Project risks

- Poor communication and proactivity
- Lack of technical skills
- Lack of time
- Significant changes in CodeDefenders during our project
- Discovering bugs in code too late
- Solar flares



Risk management

- To ensure better cooperation and productiveness of our team we will be using Scrum agile development process

- We agreed with customer to freeze current version of the project by forking the git repository so there are no interferences between our development and project maintenance by students of University of Passau.

- Commit to test strategy to discover bugs on time

Our way of working

Team's focus points

- Agile development
  - Having good communication with customer
  - Self-managing our team using SCRUM and Jira tool
- Competence buildup
  - Working together
- Test strategy
  - Analyse requirements and do testing for all use cases
  - Continuous integration, automated tests

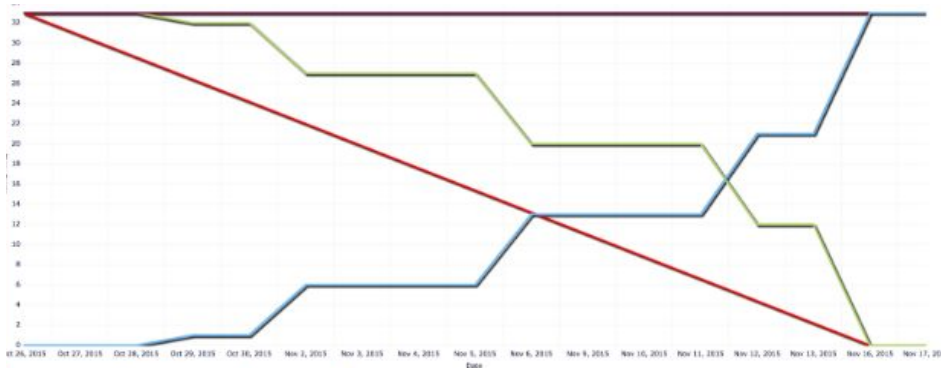
=> Quality assurance (0 Trouble Reports to customer)

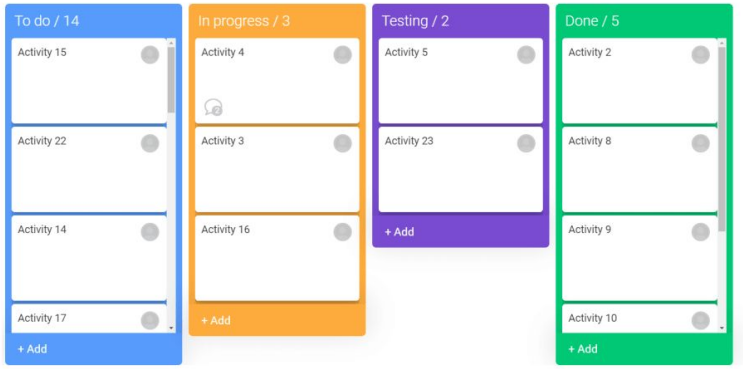
Implementation of SCRUM

- Sprints of 2 weeks
- Backlog grooming every Tuesday at 21h (CEST)
  - Sprint planning every second Tuesday after backlog grooming
- Sync meetings (daily's) 2-3 times a week (15min max)
  - Follow-up meetings if needed
- Sprint Review on Mondays after Sprint end
  - Retrospective following after the review
- Every-day communication using Discord

How will we track our progress?

- Estimating working hours needed to complete tasks
- Logging work done on each task
- Using Sprint burndown charts
- Using board to track status of tasks
- Competence matrix





Tools that we will be using

- Communication
  - Discord, Slack, Skype
- Project management
  - Jira
- Collaboration on documentation
  - Google docs, slides, drive
- Development
  - GitHub, Docker, Maven, Vagrant
  - Jenkins

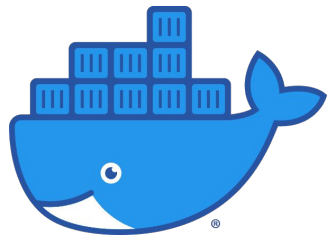




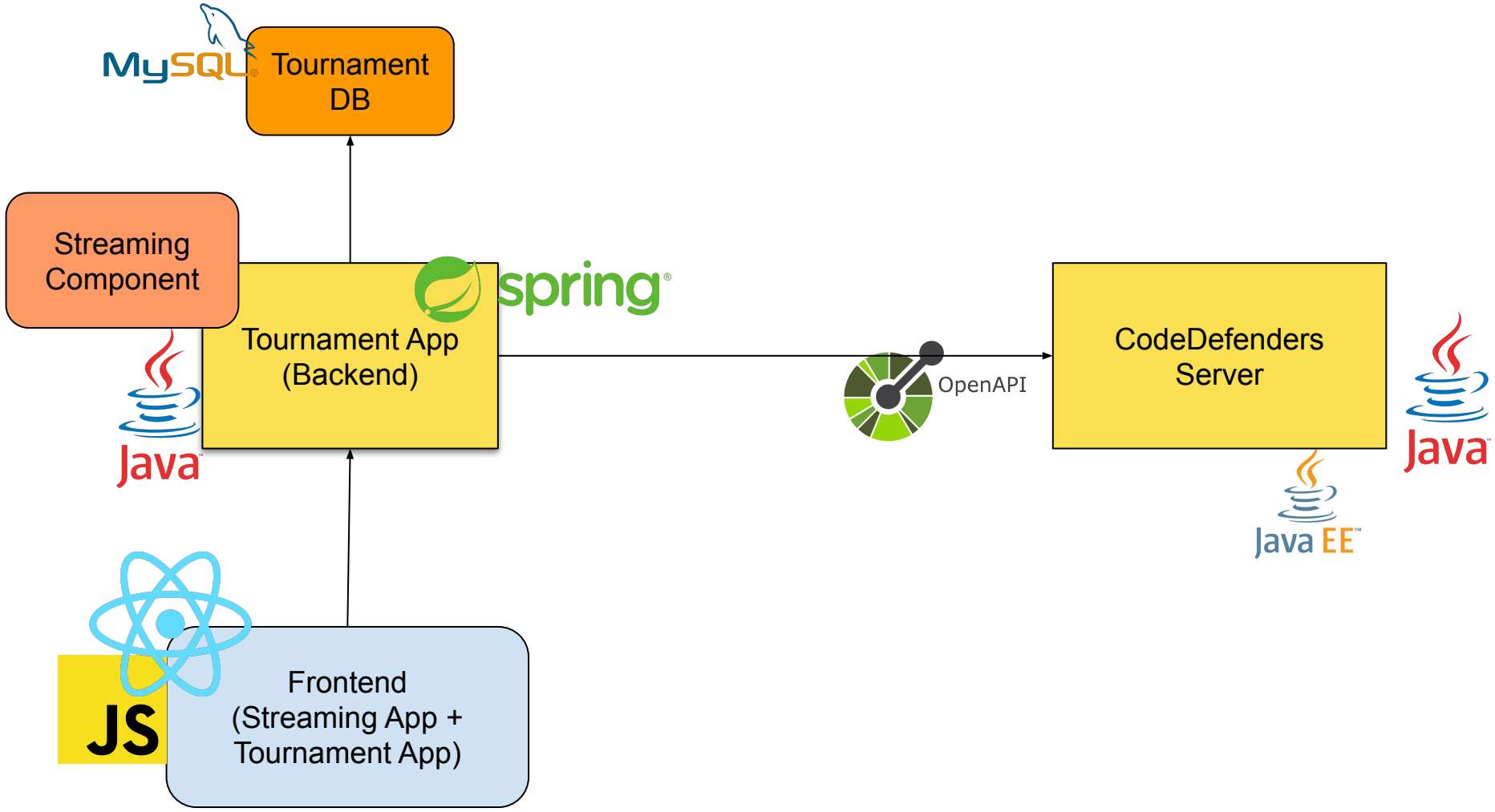









Technologies we are going to use



Motivations for technologies

Java + Spring:

- Java knowledge quite already spread among the team
- Integrates very well with concepts taken from JavaEE
- More support than JavaEE on forums and so on (also tutorials online)
- Allows to structure a Java Web application in an easy way

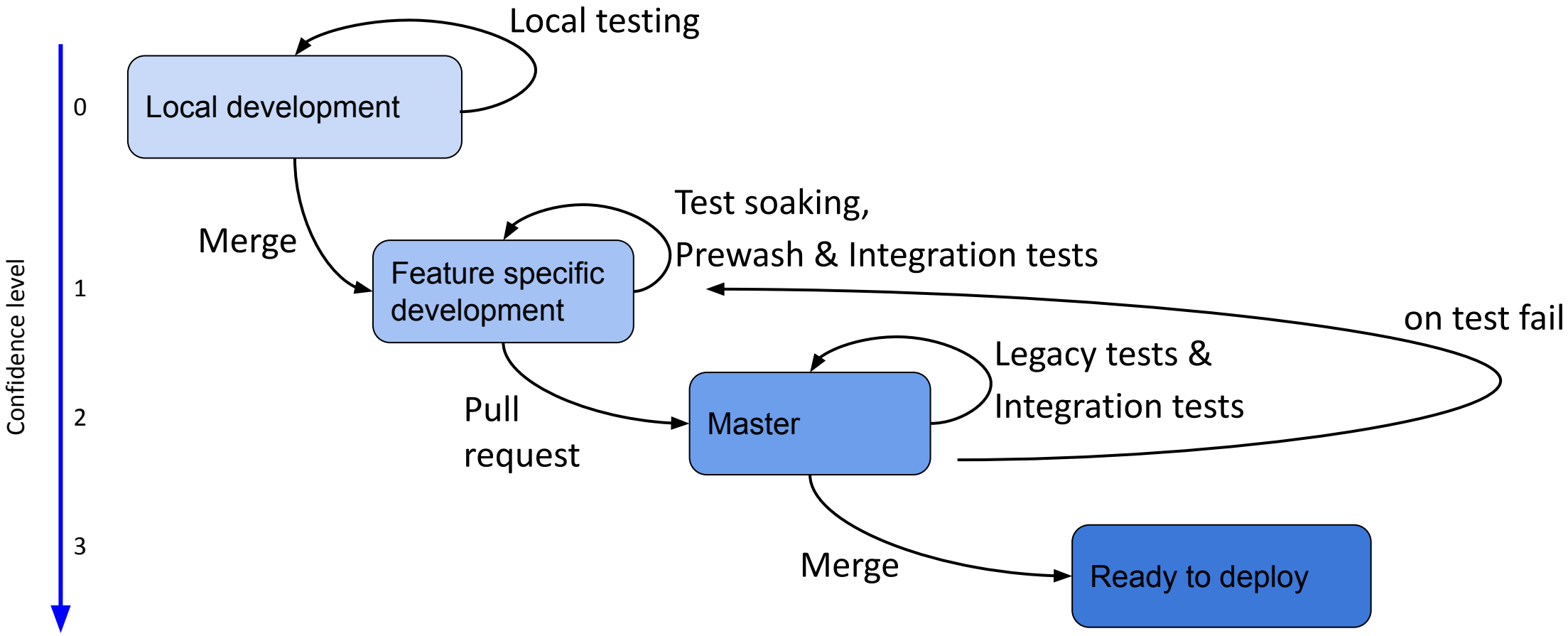
MySQL:

- The structure of the data we need will suit perfectly in a relational database.
- Team already familiar with this technology.

React:

- Free, open-source, and explanatory JavaScript library with simplistic learning curve
- Used for building simple or complex user interfaces, stable front-end framework
- Supports multi-purpose, clean architecture and platform-specific modules

Development and Testing workflow



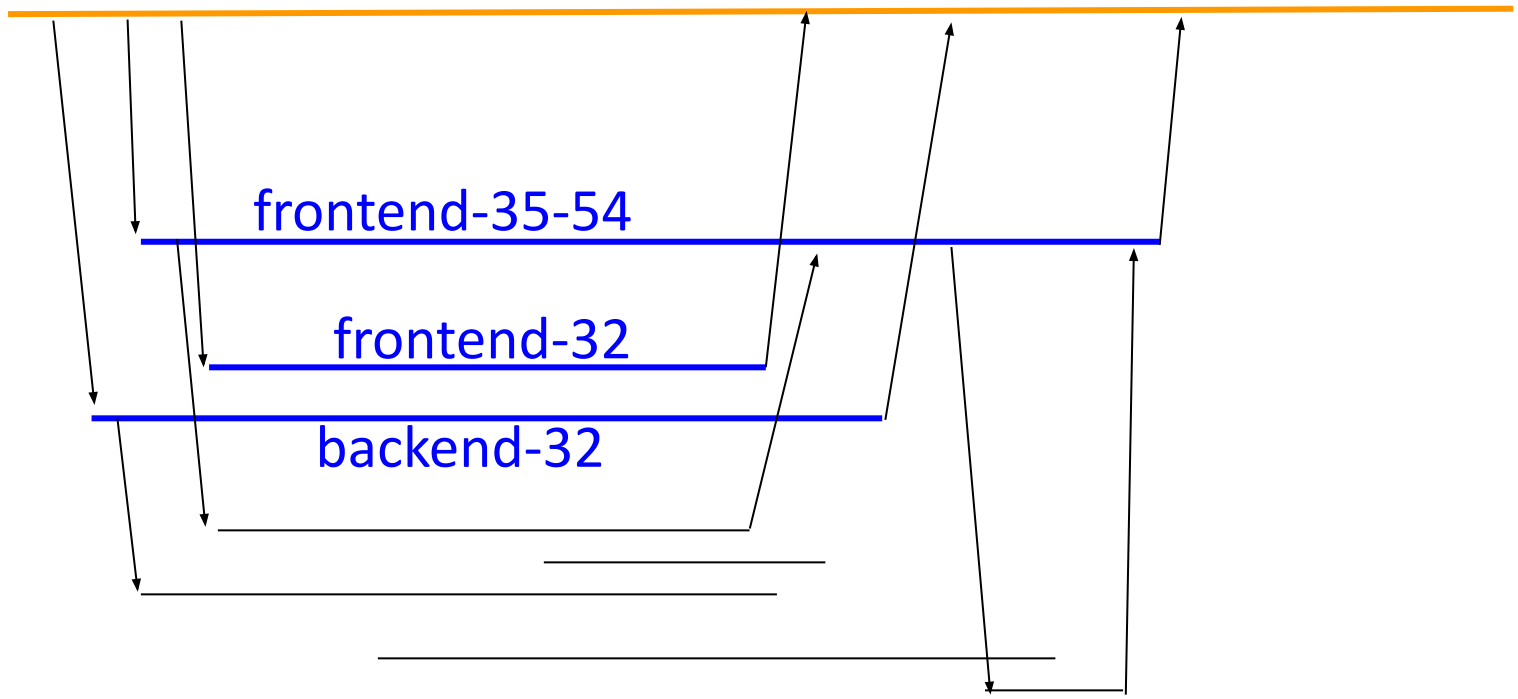
Git Branching Strategy

2 repositories -> *CodeDefenders* and *Robo-Tournament*

Master

Features development

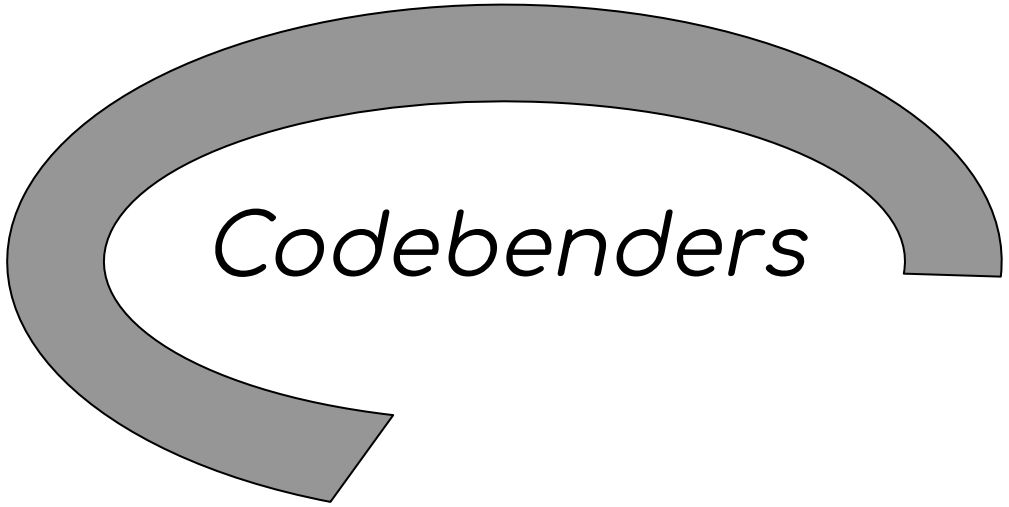
Local branches



example of Robo-Tourament repo branching

Git Branching Strategy - development workflow

1) From Master branch out feature specific development branches
   a) (one branch can be for frontend/backend of one or more user stories)
2) Team members branch out their local branches from feature specific ones
3) On local branches, developers implement smaller functional requirements and test their code locally (manually or with automated tests)
4) Once developers feel confident in their solution, they merge into that specific development branch
5) On Feature specific development branches, automated unit tests and integration tests are run after each change
   a) These are pre-wash tests (testing only the specific features in the branch)
   b) If there is need for fixes, commits are pushed directly into this branch
   c) Amend commits for further changes on the same part of code
   d) Some feature specific branches will be used for soaking tests (to test non-functional requirements)
6) When development of the features is done and all tests from the previous step confidently pass, the pull request to master can be ordered
7) Before merging into master all legacy tests are run, if any fails, pull request is denied
8) Once merged into master, integration tests are run again
   a) If there are bugs at this point, merge is reverted and development branch rebased for fixing



contact info:

fanny.delnondedieu@fer.hr
dominik.brdar@fer.hr
hrvoje.rom@fer.hr
simone.mezzaro@mail.polimi.it
fabio.patella@mail.polimi.it
andrea2.restelli@mail.polimi.it