WHAT DO YOU MEME?

Boschi Francesco, Rossi Mattia, Tumedei Gianni

Presentation Outline

- The Game
- The Lobby
- Requirements - Functional
- Requirements - Non Functional
- Design
- Game and Lobby Behavior
- Implementation
- Testing
- Continuous Integration and Deployment
- Future work
- Demo of the system

WHAT DO YOU MEME?

The Game

Like many board games before it, it’s time for “What Do You Meme” to have an online (and distributed!) counterpart. The rules are simple:

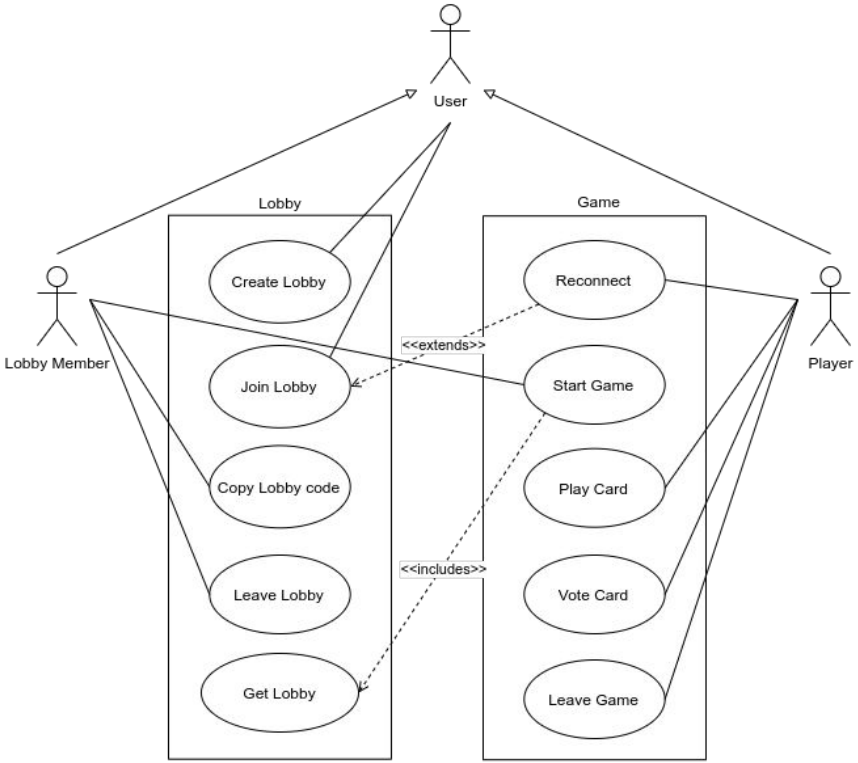
- On each turn, the judge pulls a random image card form the deck and places it on the board.
- Each player chooses, from his hand, the card with the caption that best describes the image, and places it face-down on the board.
- The judge shuffles the played cards, then flips them and votes for the best description.
- It’s now time to move on to the next round. Turns follow a clockwise direction, with a different judge each time.
- The winner is the first player to reach the maximum score.

The Lobby

In order to make What Do You Meme suitable for online play, we have devised a lobby system. The basic idea is quite straightforward:

- Players choose their username and then they can:
  - Create a new lobby and set the maximum score
  - Join an existing lobby through a code
- When inside of a lobby, players can share its code for others to join
- The lobby owner decides when to start the game

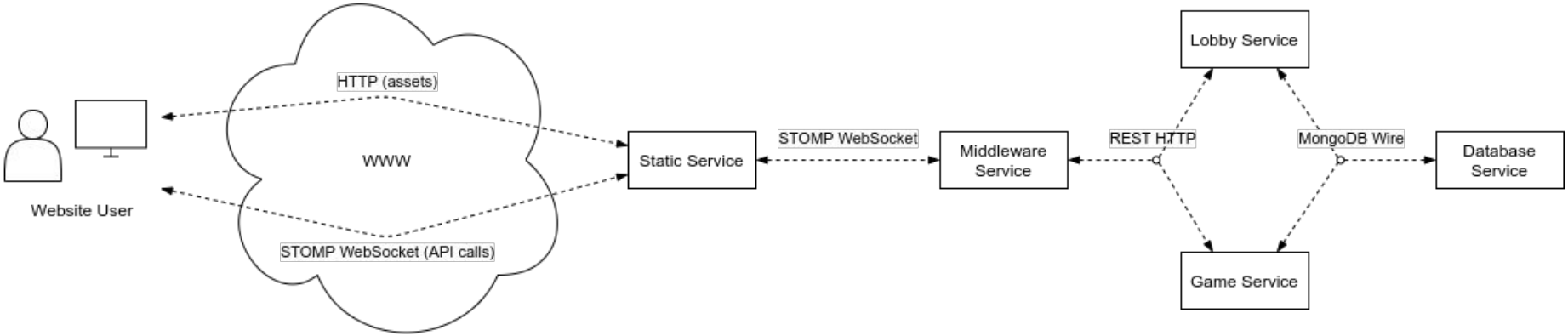
Requirements – Functional



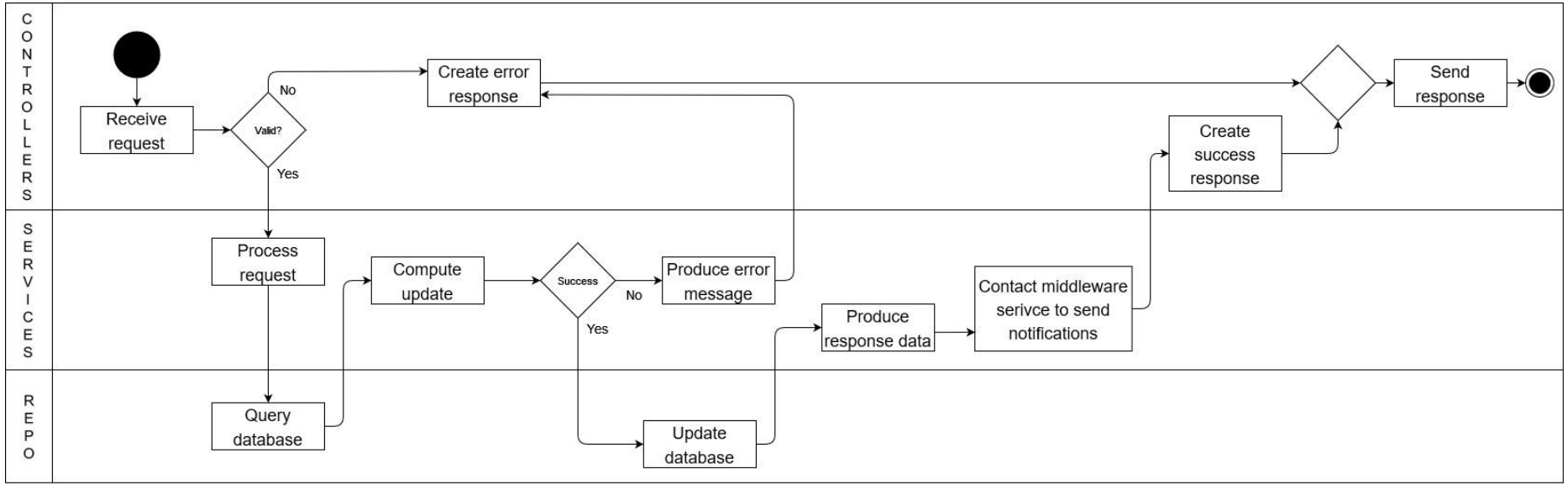
Requirements - Non Functional

- Scalability
- Resilience
- Availability
- Testability
- Ease of deployment
- Maintainability
- Code quality and readability
- Decoupling

Design


Website User
HTTP (assets)
WWW
STOMP WebSocket (API calls)
Static Service
STOMP WebSocket
Middleware Service
REST HTTP
Lobby Service
Game Service
MongoDB Wire
Database Service

Game and Lobby Microservices Behavior

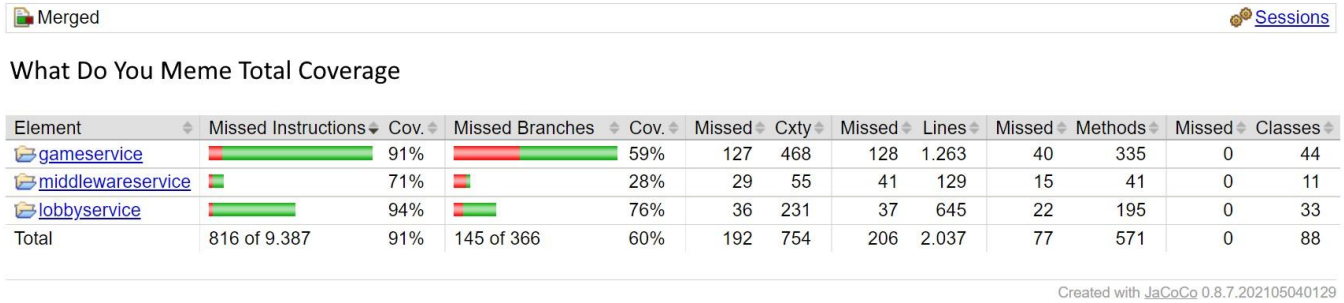


Implementation

- Microservices
  - Static Service: NGINX, Node.js
  - Middleware Service: Gradle, Spring Boot, STOMP WebSocket
  - Lobby Service, Game Service: Gradle, Spring Boot, MongoDB Repositories with transactions
  - Database Service: MongoDB with Replicas
  - Database Import Service: database seeding at startup
- Docker: ease of deployment, scalability, fault tolerance

Testing

- Unit tests on a single component of each Spring Boot microservice
  - Controller tests check the correctness of the API interface
  - Services tests check the business logic
  - Models tests check class specific features
  - External components mocked with Mockito
    - Controller tests: mock services below them
    - Service tests: mock database and HTTP requests
- 91% weighted average of the overall coverage (verified using JaCoCo)



Merged

Sessions

What Do You Meme Total Coverage

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
gameservice		91%		59%	127	468	128	1.263	40	335	0	44
middlewareservice		71%		28%	29	55	41	129	15	41	0	11
lobbyservice		94%		76%	36	231	37	645	22	195	0	33
Total	816 of 9.387	91%	145 of 366	60%	192	754	206	2.037	77	571	0	88

Created with JaCoCo 0.8.7.202105040129

Continuous Integration and Deployment

- Compilation, testing and quality assurance through GitLab CI (and locally)
- Using Docker-in-Docker service
- Two steps pipeline:
  - `compose-build`: builds and tests each container
  - `compose-up`: boots the entire system, then immediately terminates
- Automatic deployment of the documentation (compiled JavaDoc) through GitLab Pages
- Single-command deployment with Docker Compose:

```
docker-compose -f docker-compose.yml -f docker-compose.prod.yml up –build
```

Future work

- Publication of the game
- Authentication system to sign up allowing users to:
  - Set profile images
  - Interact more easily with other users (e.g., send invitations)
  - Add custom image and caption cards
  - Access a Leaderboard
- Improve the client
  - Responsive website
  - Mobile and desktop apps

Demo of the System