Veebirakenduse projekt (ITI0302)

Milestone 2

Siim Rebane

Milestone 2

- Deadline: 12th week Sunday 23:59
- 10 points, team work: one submission per team
- Written feedback, no defense
- Using LLMs is allowed
  - You have to understand everything your team submits
  - Questions about any part of the project, including a teammate's part, can be asked in the final defense
- Everything required in milestone 1 still applies: the structure, the rules for the code, the pipelines, git

1. A working process

- At least **one complete process** a user can finish end to end: from the first click to a result stored in the database and visible afterwards
  - a webstore checkout: basket → order → confirmation → the order in "my orders"
  - a booking: choose → confirm → it appears in the calendar
  - a submission and its review
- The process runs as a **logged-in user**
- Every view reads or writes through the backend; nothing is faked in the browser
- A login view; the token is sent with every request after it (lecture 8)
- Views that need a login are not reachable without it: a route guard; a `401` from the backend sends the user to the login view
- At least one table view with pagination, sorting and filtering (individual assignment 2)

2. Backend: endpoints

- At least **12 endpoints** that use the database
- Correct HTTP method (`GET`, `POST`, `PUT`, `DELETE`)
- Correct parameter type (`@RequestParam`, `@RequestBody`, `@PathVariable`)
- Endpoint names follow REST conventions
- Every endpoint serves a clear purpose
- Appropriate status codes: `201` on create, `204` on delete without a body, `404` for an unknown id, `400` for bad input
- At least one endpoint with **pagination, sorting and search/filtering** built with Specifications

2. Backend: structure

- 3-tier architecture: controller → service → repository
- Only DTOs in controllers; entities only in services and repositories
- MapStruct for DTO–entity mapping
- Lombok annotations (getters, setters, constructors, `@Slf4j`)
- JPA repositories for database operations

2. Backend: errors, validation, logging, documentation

- A **global exception handler** (`@RestControllerAdvice`, lecture 7)
  - an unknown id → `404`, bad input → `400`, with a response body that says what was wrong
  - no stack traces, SQL or table names in any response
- **Input validation** on request DTOs (`@Valid`, `@NotBlank`, `@Size`, …)
- **SLF4J** logging through `@Slf4j`; no `System.out.println`
- **OpenAPI** documentation of the endpoints: Swagger UI opens and shows every endpoint

3. Database

- At least **6 tables**
- Created and modified **only with Liquibase** changesets; no other changes allowed
- Every table has a primary key; every relation has a foreign key constraint
- **Indexes** (lecture 9): every endpoint that filters, joins or sorts has the indexes it would need with real data, as `createIndex` changesets
  - the columns in its `where`, `join` and `order by`: a foreign key, a status, a date, a search field
  - one index often serves several endpoints; a column that is only selected needs none
  - checked by reading the endpoints against the changelog; the measurement with `EXPLAIN` was individual assignment 2

4. Security

- **Working authentication** (lecture 8):
  - registration and login
  - passwords stored as bcrypt hashes, never in plain text, never logged
  - a JWT after login, checked on every request by the filter chain
  - endpoints that change data answer `401` without a valid token
- If the application has roles: `403` for the wrong role
- The JWT secret is **never in git**: it is an environment variable on the server, whether the deploy job writes it or it was set there once by hand

5. Tests

- Unit tests of services with Mockito, integration tests of endpoints on a Testcontainers PostgreSQL (lecture 7)
- **Every member** has written at least **one integration test and two unit tests** for things they developed
- **Line coverage at least 20%**, measured with IntelliJ's "Run with Coverage" or JaCoCo; the number and a screenshot in the report
- The tests run in the pipeline's build job; a merge request with a failing test is not merged

6. Server

- Backend application running in Docker on the team's server
- PostgreSQL running in a container, the data in a named volume
- Frontend built by the pipeline and served by the server's nginx (`/var/www/html` , lecture 6)
- nginx as the reverse proxy: `/api` reaches the backend
- A **domain name** assigned to the server and **HTTPS** with certbot (lab 2, lecture 6)
- A merge into `main` changes what runs on the server, with no step done by hand

7. Git and documentation

- CI/CD pipeline for the backend and for the frontend, as in milestone 1
- Feature branches, merge requests, a green pipeline and a teammate's review before every merge
- README:
  - project introduction and purpose
  - technology stack
  - prerequisites and requirements
  - how to build the project
  - how to run the application
  - how to build and run the Docker container
- Wiki: the updated database diagram

8. The report

- A wiki page in the repository, or a PDF in the submission
- What the application does now, and what changed since milestone 1: the features added, the process of part 1
- Who did what: each member's endpoints, views and tests, by name
- The test coverage: the number and a screenshot
- How to try the process: the steps, and a login the lecturer can use
- Known gaps: what does not work yet, and what is planned before the defense

Deliverable

- In Moodle, one submission per registered team:
  - the link to the repository, or both links if backend and frontend are separate
  - the `https://` address where the application opens
  - the report, or the link to it
- What is checked is the state at the deadline: the repositories, both pipelines, the application on the server, the report



Questions