Veebirakenduse projekt (ITI0302)

Individual assignment 2

Siim Rebane

Individual assignment 2

- Deadline: 11th week Sunday 23:59
- 10 points, individual work
- Written feedback, no separate defense
  - Questions about it can be asked in the final defense
  - If you cannot explain your solution, the assignment grade may be reduced or nullified
  - Theoretical questions about the subject can be asked in the test
- Using LLMs is allowed
  - Especially for explaining what is going on
  - You have to understand everything that you submit
- The analysis has to be done on code **you have written**

Where the work lives

- Either **in the team project**
  - your own entity, your own endpoint, your own view, merged through a merge request like all team work
  - it counts for the team project too
- Or **in a separate repository of your own**
  - the name starts with `iti0302-2026`, for example `iti0302-2026-<student code>-a2`; that is what gives the lecturer access
  - a Spring Boot backend with PostgreSQL in Docker and a frontend, as in assignment 1 and lecture 5
- Either way: the code is yours, and the git history shows it
- It does **not** need to be deployed; it runs on your laptop

What is checked, in one list

1. A table view: one entity of yours, listed with pagination, sorting and filtering
2. The SQL statements that view makes, each one explained
3. The N+1 problem, produced on purpose, and its fix, with the statements before and after
4. The index: the plan of the search statement before and after it, on 100 000 rows
5. The analysis document

1. The endpoint

- A `GET` list endpoint for one entity of yours
- Takes a `Pageable`: `page`, `size`, `sort`
- At least **one optional filter**: a text search, a relation (`?departmentId=`), a date or a range
  - every filter is optional, and they combine
  - built with **Specifications** (lecture 9)
- Answers a page DTO of your own: the items, the page number, the total pages, the total elements
- The entity has a **relation** to another entity: a `@ManyToOne` (an employee's department) or a `@OneToMany` (a department's employees)
- The view shows something from that relation (the department's name, the number of employees), so the endpoint has to load it
- The same relation is where the N+1 problem of part 3 is produced and fixed
- One request of the endpoint makes **two statements**: the slice and the count, no more

1. The view

- A table in the frontend (Vue, React or Angular) that shows the page
- Page buttons: next and previous, or page numbers
- Sorting by at least one column
- The filter input (or inputs)
- Every control changes the **request parameters** and asks the backend again
  - the backend sorts, filters and pages; the browser does not
- A column that comes from the relation (the department's name, not its id)

2. The statements

- Run the application with SQL logging and the bound values (lecture 9)
- One request of your endpoint with a filter and a sort
- Every statement it made, with its values put back in
- For each statement:
  - which code produced it
  - why Hibernate had to send it

3. The N+1 problem

- The endpoint of part 1, with the relation loaded lazily: one statement for the list, one more per row
  - a `@OneToMany` collection, or a lazy `@ManyToOne` read for every row
- The log that shows it: one statement for the list, one more per row
- The fix: `join fetch` in JPQL, or `@EntityGraph`
- The log after the fix
- The code change between the two logs

4. The index

- Seed the entity's table to at least **100 000 rows** on the laptop with `generate_series` (lecture 9)
  - not a Liquibase changeset, not in the pipeline
- `EXPLAIN (ANALYZE, BUFFERS)` on the search statement of part 2, with its values put back in: the plan **before**
- A `createIndex` changeset for the columns the statement filters and sorts on
- The same `EXPLAIN` **after**: the plan uses the index, the time dropped

5. The analysis document

- A PDF, 3–4 pages
- The endpoint and the view: the request with its parameters, a screenshot of the table
- The statements of part 2, each with its explanation
- The N+1 case of part 3: the log before, the code change, the log after
- The index of part 4: the plan before, the changeset, the plan after
- Everything from **your own run**, not from the lecture slides

Deliverable

- In Moodle, one submission per student:
  - the link to the repository, and to the commit or merge request that contains the work
  - the analysis PDF
  - if the work is in the team project: which entity, endpoint and view are yours
- The repository name starts with `iti0302-2026`
- The main point is to get it working and to test it through
  - It is easy to get it working with an LLM; make sure you understand what is happening
  - Questions about it in the test and in the final defense



Questions