Veebirakenduse projekt (ITI0302)

Lab 1 — Project setup

Siim Rebane

Goal of today

- By the end of this lab:
  - A Spring Boot project **runs on your machine** and answers REST calls
  - Your first **merge request** is merged in GitLab
  - You have seen **Individual assignment 1** — after today's lecture you can write all the code for it
- Deployment comes later: server + Docker next week, CI the week after
- Stuck at any step → raise your hand, that's what the lab is for

Check your tools

- Java: `java -version` → 21 or newer (latest LTS = 25)
- IntelliJ IDEA — Ultimate is free with your student licence
- Git: `git --version`
- You can log in to GitLab (gitlab.cs.taltech.ee) with your uni account

GitLab repo

- Create a project named **iti0302-2026** (visibility: private)
- Add your SSH key to your GitLab profile (Settings → SSH Keys)
  - No key yet? `ssh-keygen -t ed25519` and copy the contents of `~/.ssh/id_ed25519.pub`
  - This key is for GitLab — server keys come next week
- Clone the repo to your machine

Create the project

- IntelliJ: File → New → Project → Spring Boot (or use https://start.spring.io/ and unzip into your repo)
- Choose: Maven or Gradle (your pick), Java, latest LTS
- Dependency: **Spring Web** — nothing else needed today
- Open it, let dependencies download, run the `main` method
- You should see Tomcat start on port 8080

Hello World

```
@RestController
public class HelloController {

    @GetMapping("/")
    public String hello() {
        return "Hello ITI0302";
    }
}
```

- Rerun the application (rerun after every change)
- Open http://localhost:8080 in the browser

First merge request

- `git checkout -b feature/hello-world`
- Commit, push, open a merge request in GitLab, merge it
- This is the workflow for **everything** in this course from now on:
  - branch → commit → merge request → merge
  - Nothing goes straight to `main`

Individual assignment 1

- Full assignment deck is in Moodle — read it now, we'll walk through it
- Due: **week 5 Sunday 23:59**, 10 points
- Your variant comes from the **last digit of your student code**
- The full task: REST app + Docker on your server + CI pipeline
- Today you can already build the **code part — everything except `/api/version`**:
  - `GET /api/{resource}` — list, 200
  - `GET /api/{resource}/{id}` — 200 / 404
  - `POST /api/{resource}` — 201 + created object
  - `DELETE /api/{resource}/{id}` — 204 / 404
  - `GET /api/{resource}?name=x` — filtered list
- `GET /api/version` needs the CI pipeline — we wire it up in the week 3 lab

Pattern: in-memory storage

- No database needed for A1 — a map in memory is enough:

```
@RestController
@RequestMapping("/api/employees")
public class EmployeeController {

    private final Map<Long, Employee> storage = new HashMap<>();
    private final AtomicLong ids = new AtomicLong(1);

    @GetMapping
    public List<Employee> getAll() {
        return new ArrayList<>(storage.values());
    }
}
```

- Lab example uses `employees` — **your A1 resource is different**, adapt it
- One controller class is fine for A1; layered architecture comes later

Status codes

```
@PostMapping
@ResponseStatus(HttpStatus.CREATED)          // 201
public Employee create(@RequestBody Employee e) { … }

@GetMapping("/{id}")
public Employee get(@PathVariable Long id) {
    Employee e = storage.get(id);
    if (e == null) {
        throw new ResponseStatusException(HttpStatus.NOT_FOUND);  // 404
    }
    return e;
}
```

- DELETE: `@ResponseStatus(HttpStatus.NO_CONTENT)` → 204

Test it

- Create a file `requests.http` in your repo and run each request from IntelliJ:

```
POST http://localhost:8080/api/employees
Content-Type: application/json

{ "name": "Siim" }

###
GET http://localhost:8080/api/employees

###
GET http://localhost:8080/api/employees/999

###
DELETE http://localhost:8080/api/employees/1
```

- Check the status codes, not just the bodies — 201, 200, 404, 204

You are done when

- All A1 endpoints except `/api/version` answer correctly on your machine — right paths, right status codes, right JSON types
- Everything reached `main` through merged merge requests
- `requests.http` is in the repo, covering every endpoint
- Ahead of everyone? Read the Docker basics — next week this app goes onto a real server

Next week

- You get your own server — SSH access, Linux basics
- Bring your laptop
- **Submit your SSH public key in Moodle** (assignment "SSH key") before next week's lab
  - You already know how to make one — servers are provisioned from what's submitted
  - Submit the `.pub` file contents, never the private key