Lab 2-4

- design the solution for your problem using a CASE tool (use cases, class diagram, sequence diagram for each use case)
- use feature driven development
- layered architecture
- data validation
- all functions will be documented and tested

- use Java 8 features (lambda expressions, streams etc); the program should be written without if statements and loops
- persistence: 'in memory', text files, xml, db (jdbc); you may use any RDBMS, but we only offer support for PostgreSQL; MS SQL Server is forbidden

Lab 5: JDBC

Continue the project from Lab 2-4 - persistence in DB (using JDBC); MSSQL is forbidden.

Version 1 **(8 p)**:

- The DB repositories must implement the Repository interface from *catalog1_l1_inmemory_infile.zip* (they must not extend the InMemoryRepository class)

Version 2 **(10 p)**:

- The DB repositories must implement the *SortingRepository* interface from *catalog1-l4-sorting.zip* (they must not extend the InMemoryRepository class)
- Sorting will be implemented at the repository level (but not in the DB)
- The DB repositories will delegate sorting logic (e.g. findAll(Sort sort) method) to other components, i.e., the sorting infrastructure must be reusable
- The *getAllStudents()* method from *StudentService* (*catalog1-l4-sorting.zip*) exemplifies sorting features that must be possible using the required sorting infrastructure.

Lab 6-7: networking

- convert your project to a client-server application using sockets
- simulate an RPC server (messages sent between the client and the server must be handled in a unitary manner (simulate RPC calls))
- the server must be concurrent; use Java 8 language features; threads with ExecutorService
- on the client side, service calls must be non-blocking
- using external libraries for RPC is not allowed; only sockets
- only the db (jdbc) persistence is needed
- SOLID principles and other best practices discussed in the context of the previous project are also applied here

Lab 8: remoting

- convert your project from a client-server application using sockets to an application using RMI
- use Spring remoting
- data must be persisted to a database; use JdbcTemplate (Spring)
- use Gradle for dependency management
- xml configuration for Spring is not allowed; annotations and Java Config classes only

Lab 9: Spring, Spring Data JPA

- convert your project (Lab 2-5) to a project using Spring and JPA (Hibernate)
- only the DB persistence is needed
- use Spring --- xml config forbidden
- use Spring Data JPA (with Hibernate) --- xml config forbidden
- log messages using SLF4J
- console-based user interface
- Spring Boot is (for now) forbidden

Lab 10: REST Services

- Extend the previous solution to a modular project as follows:

1. A core module containing services, repositories, model classes
2. A web module containing controllers exposed as RESTful Web Services
3. A client module containing a console-based ui that accesses the RESTful Web Services using the RestTemplate.

Lab 11: Angular, Spring, JPA

- convert the previous project to a web application using Angular
- use Angular version 2 or higher (AngularJS/Angular1 is forbidden)
- use Spring --- xml config forbidden
- use Spring Data JPA (Hibernate) --- xml config forbidden
- log messages using SLF4J

--> for lab11: only one feature is enough

- show the list of entities (e.g. clients);
- Spring Boot is forbidden

Lab 12:

- continue the previous project
- Spring Boot is allowed, but the project structure should be the same as before (modular: core, web)
- **PART 1 (10 p):**
    - all CRUD operations
    - link entity (e.g. Rental, StudentDiscipline)
    - operations on the link entity (e.g. assign/view/etc grades)
- **PART 2 (10 p):**
    - filter, sort operations (client-side and server-side); server side versions should use features from Spring Data JPA
    - results should be paginated (using Spring)
    - data should be validated at all levels using framework (Hibernate, Spring, Angular) specific features
    - use ES6 features (or above) and follow redux principles (see *readme*)

Lab 13: handling the n + 1 select problem

- continue the previous project
- all associations *must* be lazily loaded
- after switching to Lazy fetching, check if the LazyInitializationException actually appears before trying to ‘handle' it (in SpringBoot some settings might be needed in this sense - otherwise everything is fetched eagerly)
- query the entities using: Spring Queries with Named Entity Graphs, JPQL, Criteria API, Native SQL
- in each repository (e.g: BookRepository and ClientRepository) there should be at least two methods using NamedEntityGraphs
- for each repository (e.g: BookRepository and ClientRepository), in the corresponding fragment/customized interface there should be at least two additional methods; these additional methods should have three different implementations with: JPQL, CriteriaAPI, NativeSql
- in the services only the 'main' repositories should be used (e.g: BookRepository and ClientRepository, not the fragment/customized ones)
- the application should work alternatively with all of the following configurations: EntityGraphs + JPQL, EntityGraphs + CriteriaAPI, EntityGraphs + NativeSql. The configuration switch should be possible by changing annotations or property files, but not java code.

Lab 14: testing, security, spring profiles

Testing

- Write integration tests for your repositories and services; use DbUnit, xml datasets **[1p]**
- Write unit tests for your controllers using Mockito **[2p]**

Security

- Secure the REST API; use Spring Security; passwords will be encrypted; restrict access to certain endpoints based on user roles **[2p]**
- Add a login-logout feature to your application (UI); (the rest api must be secured) **[2p]**
- In the UI, restrict access to certain routes and certain web page elements based on user roles (the rest api must be secured) **[1p]**

Spring profiles

- there should be at least two profiles (e.g. local/dev, qa, production); for example the database config could be different from one profile to another (one db for dev, a different db for prod); changing from one profile (one configuration) to another should be possible by only changing a spring profile attribute **[2p]**