All constraints

Lab 1

- The code of your project is on GitHub classroom, in a repository corresponding to your Toledo group.
- The folder "back-end/model" contains your domain model written in Typescript.
- The folder "back-end/test/model" contains all tests for your domain objects. Currently you can only test the creation of your objects, validation is for the next lesson.
- Tests are written with Jest.

Lab 2

- All layers are implemented according to the principles of layered architecture:
    - Domain
    - Services
    - Controllers
- Validation:
    - Controllers contain no validation.
    - Services contain overarching validation rules.
    - Domain objects contain input validation and business validation specific to that domain object.
- Testing:
    - All domain objects are fully tested with Jest, including validation.
    - All services are fully tested with Jest.
    - Controllers are tested via swagger (manually), no separate tests required.
- The data coming in via requests in the router is encapsulated in Data Transfer Objects. These DTO's are defined in a file **index.ts** in the **types** folder.
- All routes are fully documented and executable with Swagger via the url **/api-docs**.
- For each type, there is a fully worked-out component schema defined at the top of the controller itself.

Lab 3

- A Next.js front-end app is installed in the front-end directory.
- All pages that require a route are placed in the "pages" folder.
- Pages are built from several reusable components placed in the "components" folder.
- Components are not implemented directly in a page.
- "Props" are used to display dynamic content within components.
- "State" is used to store information between different renders of a component (no local variables!).
- Callback functions are used to notify higher-level components or pages of an event within the current component.
- Calling a Rest API is done in separate, reusable Services. No fetch logic is ever written directly into a component.
- Dynamic routing should be used in appropriate places.
- You use events in different places (onclick, onhover, ...).

Lab 4

- You no longer work with static data in the repositories, but with a real database.
- A local postgreSQL database is installed and used. The configuration to connect is in an **.env** file.
- Prisma is used as ORM.
- Changes to existing entities must be done in the service layer through the domain model and only then send to the database via the repository layer. This way, business and validation rules are not violated.
- The database schema is modelled in a Prisma schema and the Prisma client is generated.
- Database objects in the domain layer use the Prisma client to query the database.
- No Prisma objects are passed to the service layer. Prisma objects are mapped to domain objects in a static **from** method of the corresponding domain object.
- At least 1 one-to-many relationship is modelled.
- At least 1 many-to-many relationship is modelled in your prisma schema and domain objects. No circular dependency should exist in your domain layer, so decide whether to use an intermediate object or to make the relationship in the domain uni-directional.

Lab 5

- The hook useSWR is used for API requests.
    - SSR and SSG can be applied optionally.
- The hook useEffect is used for interaction with an external system (e.g. browser storage).
- There is at least 1 functional form with validation, error handling and integration with the back-end.
- There is at least 1 login form with validation and error handling.
- At least 2 values are stored in browser storage and used throughout the application.
- Styling is applied to the extent that your application is usable and readable. You may choose your own styling framework.

Lab 6

User Sign-Up

- Passwords are always encrypted with bcrypt when stored in the database.
- User input is always validated (back-end and front-end).

Authentication

- You use JWT token-based authentication where appropriate in routes and Swagger.
- Except for login, register, status, the Swagger documentation and possibly a limited number of other routes depending on the context of your project.

Authorisation

- You have at least 3 different roles in your domain.
- At least 1 route has different behaviour depending on the role (gives different data based on the role).

Lab 7

Front-End Security

- You can log in, log out and register users.
- Token-based authentication with JWT is used.
- Authentication: most pages are protected.
- If the user calls up a functionality/page he is not authorised to access, the user is properly informed.
- Authorisation: at least 1 page provides different content based on the role.
- On the homepage a table with a number of predefined users is provided, that we lecturers can use to test your project. For example:

username	password	role
user1	user1	role in the context of your project (eg. lecturer)
user2	user2	role in the context of your project (eg. student)
user3	user3	role in the context of your project (eg. guest)
user4	user4	role in the context of your project (eg. admin)

i18n

- You can show at least 1 page of your project in at least 2 languages/locales.
- You can switch languages in a user-friendly way on each of those pages.

Lab 8

Helmet

- Your HTTP traffic is secured with Helmet.

React Testing Library

- At least 2 components are tested with React Testing Library.