# Full-Stack: Final project requirements 2024-2025

## General

1. The project is versioned in a GitHub repository which is initialized via GitHub classroom.
2. A release tag with the name “FINAL” is created before the deadline of the assignment. This tag will be reviewed.
3. The project consists of an "analysis", "front-end" and "back-end" folder.
4. The project follows the same structure as the provided start template.
5. The login page contains a table of predefined users that can be used for testing, e.g.

| **Username** | **Password** | **Role** |
|---|---|---|
| user1 | user1 | … |
| user2 | user2 | … |

6. Your project contains a seed.ts containing test data that we can use to test your stories. This script must be executable via "npx ts-node util/seed.ts".
7. We need to be able to start your project immediately with our own database configuration by using the steps:
    a. Back-end: npm install; npx prisma migrate dev; npm start
    b. Front-end: npm install; npm start
8. Create a movie (max. 4 minutes) to demo the project.

## Functional

1. All the user stories work as described in your analysis and demonstrate that you have mastered all technical expectations. Your analysis is up to date with your code.

## Domain

1. The domain model contains at least 4 entities and reflects the domain model in the analysis.
2. There is at least 1 one-to-many and 1 many-to-many relationship in the database.
3. Many-to-many relationships in the database are not bi-directionally mapped in the domain.
4. The database schema is modelled by means of a Prisma schema (in *repository/prisma*).
5. A local postgres database is used, connection configuration is stored in a separate .env file.
6. CRUD operations are executed via Prisma client objects.
7. Layered architecture:
    a. Domain objects only depend on other domain objects (not repositories, services or controllers)
    b. Prisma client objects are mapped to domain objects via a static *from* function in the domain object itself.
    c. No Prisma client objects are passed to services or controller
8. Domain objects contain input validation and business rules validation.
9. Testing: Domain objects are fully tested with Jest.

## Services

1. Services only do orchestration and do not contain domain specific business logic.
2. Services only contain overarching validation rules that can't be handled by the domain.
3. Updates on existing entities are first validated using the domain model before calling the database layer.
4. Layered architecture: services only delegate to the domain or other services, not to controllers.

5. Testing: all services are fully tested with Jest. Mock objects (jest.fn()) are used to ensure isolated testing.

## Controllers

1. All API routes are housed in controllers that are structured and split according to use-case or entity.
2. Controllers only delegate to services and do error handling. They should not contain any other functionality.
3. At least one route exists for every HTTP operation (POST, PUT, GET, DELETE).
4. Each API route is fully documented and testable with Swagger on the url /api-docs.
5. For each known input- and return type, there is a fully detailed component schema defined at the top of the controller itself.
6. Incoming data in the routers' endpoints is encapsulated in Data Transfer Objects. These DTO's are defined in *types/index.ts*.

## Front-end

1. All routes in the front-end that a user can navigate to have a page in the “pages” folder.
2. There is at least 1 dynamic route.
3. A page only contains a skeleton and is composed of reusable React components.
4. Fetching logic for calling the backend API is encapsulated in separate reusable services.
5. At least one form with a minimum of 2 fields (besides the user login form) exists.
6. A user login form exists.
7. Every form contains validation, error handling and event handling.
8. There are at least 1 GET, 2 POST, 1 PUT and 1 DELETE requests to the back-end API which are fully implemented in the back-end.
9. At least two values are stored in browser session storage and used across the application.
10. Hooks are used in the correct way: useSWR for API requests, useEffect for interaction with external systems, useInterval for polling.
11. I18n:
    a. The application is available in 2 languages (min. 1 page).
    b. There is a user-friendly way to change the language of these pages.
12. Testing: At least 2 components are tested using React Testing Library.

## Security

1. Passwords are stored encrypted in the database using bcrypt.
2. Token-based authentication with JWT is used.
3. Login, user sign-up, api-docs are excluded from authentication. All other pages are protected with authentication.
4. It is possible to authenticate via swagger and test protected endpoints.
5. There exist at least 3 roles in the domain.
6. Back-end: at least 1 route shows different behavior depending on the role.
7. Front-end: at least 1 page shows different behavior/data depending on the role.
8. Authentication/authorization errors are correctly displayed in the front-end.
9. HTTP traffic is protected using Helmet.