Exam-Management-System Maintenance Guide

Introduction and Motivation

The Exam-Management-System is designed for the creation, management, and improving of exams.

By given web interface, inspection and maintenance of exams become faster easier, and more efficient.

The motivation behind this system is to reduce the manual workload, minimize errors, improve quality of the questions, and improve the overall exam experience for both educators and students.

Maintenance:

Language: JavaScript

Exam-Management-System is implemented using JavaScript only.

Server-Side Maintenance:

Server: Restify

Restify is used as the server framework to handle API requests.

The server configuration, api-endpoints and middleware are all define at: src/main/server.js

For more information about restify

Authentication: JWT

JSON Web Tokens (JWT) are used for securing the API endpoints.

The jwt is transform to the client on login, and on client request (see /refreshJWT endpoint).

If within api request valid jwt token is exist, The server will add to each request body an object: “callingUser” with the jwt information, if the jwt is not valid, the request will be failed (except for /login, /logout /changePassword)

Database: PostgreSQL

The system is configured to work with postgres.

Database Communication: Sequelize

Sequelize is an ORM for interacting with the PostgreSQL database.

For the schemas definitions head to the src/main/DAL, the repositories and data-objects will be defined there.

For more information sequelize

Common Maintenance and Development Use Cases

1. **Running the Server**
   - **Install dependencies:**
   - `Cd src/main`
   - `Npm install`
   - `Npm run dev`

2. **Adding/Maintaining API Endpoints**
   - Add endpoint in src/main/server.js

```
      server.post('/signUp', service.signUp);
```

   - And handlers in src/main/business/applicationFacade.js

```
      async register(userdetails){...}
```

   - example
3. **Adding/Maintaining Models**
   - Define Sequelize models in the `src/main/DAL` directory.

```
      function defineUserModel(sequelize) {...}
```

4. **Authentication Implementation**
   - Adding information to the JWT: in the signIn handler

5. **Compile text to latex:**
   - Different object need to be compile to latex, this could be done using the LatexCompiler, for example:

```
compileNormal(answer.text, this.#createCropCallback('5 5 5 5',
callback));
```

   -

6. **Testing Backend**
    - **Run tests:**

      ```
      Cd src
      npm run test
      ```

Client-Side Maintenance

Frontend: React JS

The frontend of the Exam-Management-System is developed using React JS.

The react is using .next framework, for more information docs

Common Maintenance and Development Use Cases

1. **Running the Client**
    - **Install dependencies:**

      ```
      cd ReactApp
      npm install
      ```

    - **Start the client:**

      ```
      cd ReactApp
      npm install
      npm run dev
      ```

2. **Adding/Maintaining Pages**
    - Define new pages and components in the /ReactApp/src/page, Due to .next, the page will be automatically added.
    - Add new component to the sidebar, via /ReactApp/src/layout/dashbord/config.js as follow:

```
{
    title: TITLE,
    path: '/page-path,
    icon: (
      <SvgIcon fontSize="small">
        <Newspaper />
      </SvgIcon>
    ),
    permissions: [types.WHO_HAS_PERMISSION]
  }
```

   -

13. **Network Requests**
    - Use The following functions from /ReactApp/src/utils/rest-api-call/js

```
export async function requestLatexServer(path, body) {...}
export async function requestServer(path, method, body) {...}
```

Those function are connected to the server and will do the rest-api call

---

PDF Generation using LaTeX

The system compiles exams into PDF format using a LaTeX server.

LaTeX server implementation: The latex server is based on functional programming, using callbacks to build the final exam pdf.

The main function is

```
compileNormal(latexCode, callback) {...}
```

This enable to add new "latex object" to the system and dynamically build the exam.

Possible Future Features

- Add more task types
- Support Hebrew in the latex format.
- Grading student exams
- Statistical analyse of student answers
- Statistical analyse of keys/distractor s

This guide provides a overview of the maintenance and development processes for the Exam-Management-System. For detailed implementation and further customization, refer to the respective documentation links provided.