ExamLendOut backend API

This Repository contains the Database logics and Backend api for an `ExamLendOut` -Service.

The individual modules are executed as docker containers.

Status

`Currently the System is ready for interacting with the frontend . though it is still in dev mode`

Database schema

- the database *examlendoutDB* contains a total of 4 tables `user` , `exam` , `solution` , `summary` maintaining the following schema :

examlendoutDB

- the schema.sql is located in the database root folder with the name initneu.sql

description

- a **user** can create an **exam** or more.
- a **user** can create a **solution** or more to an existing **exam**.
- a **user** can create a **summary** or more to an existing **exam**.
- an **exam** could exist only if a **user** exist.
- **solution** and **summary** both could exist only if a **user** `and` **exam** exist.
- when `DELETE` or `UPDATE` an **exam** tables **solution** and **summary** will `CASCADE` .
- when `DELETE` or `UPDATE` a **user** tables **exam**, **solution** and **summary** will `CASCADE` .

Backend API

the project uses :

- **express** for http req.

- **mysql** for database connection.
- **cors** for cors for cross-origin resource sharing
- **cookie-parser** for parsing cookies
- **dotenv** to access environment variables from .env for thr database connection

Project description :

- the **root** folder contains `app.js` for parsing reqests and use routes for handling api, and the entry-point `server.js` is listening on port:

> :8080

- the folder **Model** contains the file `DB` where the mysql client create a connection to the **examlendoutDB** .
- the folder **routes** contains files :
  `router` , `elouser` , `eloexam` , `elosummary` , `elosolution` , `index`
- `router` define an API for handling various routes.
- `elouser` , `eloexam` , `elosummary` , `elosolution` each provide 3 express http requests (get,post,delete) to the database which implement the mysql CRUD operations .
- `index` provide a hello world message

how to use ?

- **get http://localhost:8080/ => index => hello world**
- **get http://localhost:8080/user => shows the content of the user table**
- **post http://localhost:8080/user => add a new user**
- **delete http://localhost:8080/user/:id => delete a user**
- **get http://localhost:8080/exam => shows the content of the exam table**
- **post http://localhost:8080/exam => add a new exam**
- **delete http://localhost:8080/exam/:id => delete an exam**
- **get http://localhost:8080/solution => shows the content of the solution table**
- **post http://localhost:8080/solution => add a new solution**
- **delete http://localhost:8080/solution/:id => delete a solution**
- **get http://localhost:8080/summary => shows the content of the summary table**

- **post http://localhost:8080/summary => add a new summary**
- **delete http://localhost:8080/summary/:id => delete a summary**

`new routes`

- **get http://localhost:8080/user/solbyuser/:id => get solution data and id from a chosen user table**
- **get http://localhost:8080/user/sumbyuser/:id => get summary data and id from a chosen user table**
- **get http://localhost:8080/user/exambyuser/:id => get exam data and id from a chosen user table**
- **get http://localhost:8080/exam/solbyexam/:id => get solution data and id from a chosen exam table**
- **get http://localhost:8080/exam/sumbyexam/:id => get summary data and id from a chosen exam table**
- **get http://localhost:8080/user/:id => shows the content of the chosen user table**
- **get http://localhost:8080/exam/:id => shows the content of the chosen exam table**
- **get http://localhost:8080/solution/:id => shows the content of the chosen solution table**
- **get http://localhost:8080/summary/:id => shows the content of the chosen summary table**

`searching`

1. get every thing from exam in descending order (get the last created at first) `the result could be a little complex because of th 1-n relationship lead to more programming in frontend`

- send query data :

```
{

    "semester": "2",
    "prof": "prof",
    "module": "module",
    "amount": 5
}
```

- to :

- **http://localhost:8080/exam/full=> shows the content of all related exam with all related tables sol and sum**
- and get :


```
{
    "idm": 1,
    "dataex": "testdata",
    "module": "module",
    "prof": "prof",
    "CP": null,
    "aid": 1,
    "notes": null,
    "semester": "2",
    "idy": 1,
    "datasu": "assets/sum.pdf",
    "idn": 1,
    "dataso": "assets/sol.pdf"
}
```


2. get only exam data in descending order (get the last created at first) `this is the RESTfull way (ressource from one table per path ) and you must less programming in frontend`

- send query data :


```
{

    "semester": "2",
    "prof": "prof",
    "module": "module",
    "amount": 5
}
```


- to :
- **http://localhost:8080/exam/fullbetter=> shows the content of all related exam without related tables sol and sum**
- and get :


```
{
    "idm": 1,
    "dataex": "testdata",
    "module": "module",
    "prof": "prof",
    "CP": null,
    "aid": 1,
    "notes": null,
```

```
    "semester": "2"
}
```

- with idm you can get all sum and sol in their corresponding path ;)

authentication

1. get user id

- send query data :

```
{
    "name": "usrtest",
    "password": "pwd"
}
```

- to :
- **http://localhost:8080/user/authentication**
- and get :

```
{

    "idr": 1

}
```

all routes are tested with the postman

Docker-compose modifications

in db container

```
volumes:
     - ./database:/usr/src/app:rw
     - ./database/initneu.sql:/docker-entrypoint-initdb.d/initneu.sql:ro
```

- first line to persist data between host and container
- second line for creating the database the first time we run the container

in backend container

```
depends_on:
      - "db"
```

- due to the asynchronous starting of containers in docker-compose this option should control the order of service startup and shutdown so the backend will start after the database start .
- `this will not help when running the docker-compose first time because creating the db takes longer time than getting the container itself ready`

Configuration

You have to create your own .env files in the directories `/backend & database` and configure them according to the already existing .env.example templates.

starting containers the first time

```
docker-compose up
```

- this will start the backend and the database container simultaneously .
- the database container will fetch the database schema into the mysql server to create the database `examlendout` only in the first build and run of the container.
- for the next runs the data will still persisted between the host and the container that means that the database will not created every time the container runs and every pushed data will still in the database after a shutdown or restart of the container .
- in the first run of the conatiners the database container will take more time to create the database than to be ready for hosting this will block the backend from getting data on the first run though the index still be reachable at http://localhost:8080/ .
- `after the first run of the containers the backend should be restarted one time to be able to get data from the database` I do this simply by stoping containers with Ctrl+c and restart them with docker-compose up .
- after that you should get a test user from http://localhost:8080/user

how to send data to the backend

- To post a `user`

```
{
    "name": "test",
    "password": "test",
    "permissionlevel": 1
}
```

- To post an `exam`

```
{
    "dataex": "path/path/",
    "semester": "2",
    "prof": "prof",
    "CP": 2,
    "module": "module",
    "aid": 1,
    "notes": "Notes",
    "usrid": 1
}
```

- To post a `solution`

```
{
    "dataso": "path/path/",
    "id_user": 1,
    "id_exam":1
}
```

- To post a `summary`

```
{
    "datasu": "path/path/",
    "id_user": 1,
    "id_exam":1
}
```

tests

`in Dev`

Develope

Here for you have to build the docker container with the `docker-compose.yml` file.

1. You must allow Docker to access this directory on your host machine. You can set this option the `File Sharing` -Preference.
2. Run the `docker-compose.yml` file.

3. Now you are able to access the backend directly on the port set in your .env file in the `/backend` folder