Matteo CHIANALE
Amine Sho

Soudaisse HADJI

Contents

API FRONTEND....2
DATABASE & DATA....2
JOIN DATABASE TO API....2
POST DATA....3
API....4
Setup and Middleware Integration....4
Database Configuration....4
API Endpoints and Logic....4
Development Approach....5

API FRONTEND

DATABASE & DATA

- Using PgAdmin4, with PostGreSQL langage
- Generate. json data for learning packages & learning facts.

JOIN DATABASE TO API

- Using sequelize
- Defined our tables models in **.ts in models.ts**
- Example User table:

```
const User = sequelize.define('userfit', {
    username: {
        type: DataTypes.STRING,
        primaryKey: true,
    },
    mail: {
        type: DataTypes.STRING,
        allowNull: false,
    },
    firstname: {
        type: DataTypes.STRING,
        allowNull: false,
    },
    lastname: {
        type: DataTypes.STRING,
        allowNull: false,
    },
    mdp: {
        type: DataTypes.STRING,
        allowNull: false,
    },
    token: {
        type: DataTypes.STRING,
        allowNull: false,
        defaultValue: '', // Set the default value to an empty string
    },
}, {timestamps: false});
```

- Defined relations:

```
//Relations
User.hasMany(LearningPackage, options: { as: 'user_package', foreignKey: 'username', onDelete: 'CASCADE'  });
LearningPackage.belongsTo(User, options: { foreignKey: 'username' });

LearningPackage.hasMany(LearningFact, options: { as: 'package_fact', foreignKey: 'id_package', onDelete: 'CASCADE'  });
LearningFact.belongsTo(LearningPackage, options: { foreignKey: 'id_package' });

User.hasMany(LearningEvent, options: { as: 'user_event', foreignKey: 'username', onDelete: 'CASCADE'  });
LearningEvent.belongsTo(User, options: { foreignKey: 'username' });

User.belongsToMany(User, options: { as: 'Followings', through: Follower, foreignKey: 'id_follower', otherKey: 'id_following' });
User.belongsToMany(User, options: { as: 'Followers', through: Follower, foreignKey: 'id_following', otherKey: 'id_follower' });

User.belongsToMany(LearningPackage, options: { as: 'LikedPackages', through: PackageLike, foreignKey: 'liker', otherKey: 'like_package' });
LearningPackage.belongsToMany(User, options: { as: 'LikedBy', through: PackageLike, foreignKey: 'like_package', otherKey: 'liker' });

User.belongsToMany(LearningPackage, options: { as: 'learnPacks', through: PackLearn, foreignKey: 'learner', otherKey: 'pack_learned' });
LearningPackage.belongsToMany(User, options: { as: 'LearnedBy', through: PackLearn, foreignKey: 'pack_learned', otherKey: 'learner' });

export {User, LearningPackage, LearningFact, LearningEvent, Follower, PackageLike, PackLearn}
```

(We don't use all relations, perhaps we want to create a social system with other Users). We separated what packages users learned from packages created by users.

POST DATA

- Post_data.ts, reading json file and using sequelize functions bulkCreate to insert a lot of values in just one function.

```
const sequelize = require('./sequelize-config.ts');
import  {User, LearningEvent, LearningPackage, LearningFact ,PackLearn  } from './models';
const fs = require('fs');
const { promisify } = require('util');
const readFileAsync = promisify(fs.readFile);
sequelize.sync({ force: true }).then(async () : Promise<void>  => {
    //User
    const users =  await User.bulkCreate( records: [
        {username: 'user0', mail: 'user0@gmail.com', firstname: 'M.', lastname: 'Toto', mdp: 'Test'},
        {username: 'mister design', mail: 'mister design@gmail.com', firstname: 'M.', lastname: 'Design', mdp: 'Test'},
        {username: 'amine', mail: 'amine@gmail.com', firstname: 'Amine', lastname: 'Sho', mdp: 'Test'},
        {username: 'mchianale', mail: 'matteo.chianale75@gmail.com', firstname: 'Matteo', lastname: 'Chianale', mdp: 'Test'},
        {username: 'mr_learning', mail: 'mr_learning@gmail.com', firstname: 'Mr', lastname: 'Learning', mdp: 'Test'},

    ])
    const jsonData = await readFileAsync( arg1: 'learning_packages.json', 'utf8');
    const packagesData = JSON.parse(jsonData);
    const jsonData2 = await readFileAsync( arg1: 'facts_data.json', 'utf8');
    const factsData = JSON.parse(jsonData2);

    await sequelize.sync();
    await LearningPackage.bulkCreate(packagesData);
    await LearningFact.bulkCreate(factsData)

    console.log('Data inserted successfully.');
});
```

API

Setup and Middleware Integration

Express & Middlewares: The application is set up with Express, a popular Node.js framework. You've used body-parser for parsing incoming request bodies and cors to enable Cross-Origin Resource Sharing, which is crucial for web applications interacting with this server from different domains.

Database Configuration

Sequelize: The ORM (Object-Relational Mapping) tool Sequelize is used for database interactions. This setup abstracts SQL queries, making it easier to work with your database using JavaScript objects and functions.

API Endpoints and Logic

1. Liveness Check:

- /api/liveness: A simple GET request to check if the API is live.
- User Authentication:
- /api/login/:mail/:mdp: A PUT request for user login. It checks if the user exists and if the password is correct, then generates and returns a token.
- /api/disconnect/:token: A PUT request to log out a user, essentially clearing their session token.
- /api/register: A POST request to register a new user. It includes validations for email, password, firstname, and lastname.

2. User Information Retrieval:

- /api/informations/:token: A GET request to fetch user information like followers and following data, using the session token.
- Learning Package Management:
- /api/packages: A GET request to fetch all public learning packages.
- /api/packages/search_by_filters/:token: A POST request to fetch packages based on various filters.
- /api/package_by_id/:id_package: A GET request to fetch a specific learning package by its ID.
- /api/:token/:id_package: A GET request to check if the current user owns a specific package.
- /api/:id_package: A GET request to fetch facts related to a specific package.
- /api/learned/:token/:id_package/:check: A POST request to mark a package as learned by a user.
- /api/:token/new_package: A POST request to create a new learning package.
- /api/update/:token/:id_package: A PUT request to update a learning package.
- /api/:id_package/new_fact: A POST request to create a new fact related to a learning package.

3. User-Specific Data:

- /api/user/:token/get_username: A GET request to retrieve the username of the current user.
- /api/user/get_information/:username: A GET request to retrieve specific information about a user, like the total number of learned packages, created packages, and average difficulty level of learned packages.

Development Approach

- Modular and RESTful Design: The API follows RESTful principles, making it scalable and maintainable. Each endpoint serves a specific purpose, adhering to the HTTP verb semantics (GET, POST, PUT).
- Validation and Error Handling: You've incorporated input validation for email, password, and other fields to ensure data integrity. Appropriate error handling is in place to respond with relevant messages and status codes.
- Token-Based Authentication: User sessions are managed using tokens, which is a secure and stateless way to handle authentication.
- ORM for Database Interactions: Using Sequelize simplifies database operations and reduces the risk of SQL injection attacks.
- Asynchronous Programming: The use of async/await makes the code dealing with asynchronous operations like database queries more readable and easier to manage.

Our tokenization system during Login, Register and Logout:

```
//LOGIN
app.put( path: '/api/login/:mail/:mdp', handlers: async (req : Request<{mail: string} & {mdp:... , res : Response<any, Record<string, a... ) : Promise<Response<...>>  => {
    const mail : string  =  req.params.mail
    const mdp : string  = req.params.mdp
    const users = await User.findAll();
    const current_user = users.find((U) : boolean  => U.mail === mail);
    if (!NotNull(current_user)) {return res.status( code: 400).send( body: {message: `User mail doesn't exists ${mail}`});}
    if (current_user.mdp === mdp){
        const token : string  = Math.random().toString( radix: 36).substring(2) + Date.now().toString( radix: 36); //Token
        //Update token
        current_user.token = token
        await current_user.save()
        return res.status( code: 201).send( body: { token });
    }
    else {
        return res.status( code: 400).send( body: {message:'Invalid Password'});
    }

});
```

FRONTEND

Video demo