Server (Express, Sequelize)

Prerequisites

- Node JS
- PostgreSQL
  - username
  - password

Setup

- CLI

```
mkdir express-sequelize && cd $_ && touch {index.js,sequel
ize.txt};

npm init -y && git init;
npm install pg express sequelize && npm install sequelize-
cli nodemon --save-dev && echo "node_modules" > .gitignor
e;
npx sequelize-cli init;

npm install bcrypt jsonwebtoken dotenv;
npm install jest supertest --save-dev;
```

- Git (initial)

```
git add . && git commit -am "initial"
```

Update scripts

- package.json scripts

```
{
  "name": "express-sequelize",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "nodemon index.js",
    "test": "npx jest --runInBand --forceExit --detectOpenHa
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "jsonwebtoken": "^9.0.2",
    "pg": "^8.11.5",
    "sequelize": "^6.37.3"
  },
  "devDependencies": {
    "jest": "^29.7.0",
    "nodemon": "^3.1.0",
    "sequelize-cli": "^6.6.2",
    "supertest": "^7.0.0"
  }
}
```

Create Database

- Change the config

```
{
  "development": {
```

```
    "username": "postgres",
    "password": "postgres",
    "database": "notes_sequelize_express",
    "host": "127.0.0.1",
    "dialect": "postgres"
  },
  "test": {
    "username": "postgres",
    "password": "postgres",
    "database": "notes_sequelize_express_test",
    "host": "127.0.0.1",
    "dialect": "postgres"
  },
  "production": {
    "username": "root",
    "password": null,
    "database": "database_production",
    "host": "127.0.0.1",
    "dialect": "mysql"
  }
}
```

- Create database

```
npx sequelize-cli db:create
```

Migrations, Models, Seeders

- Create model (add the command to sequelize.txt)

```
# order matter (generate the one without the Foreign Key f
irst)
npx sequelize-cli model:generate --name User --attributes
name:string,age:integer,email:string,password:string
```

```
npx sequelize-cli model:generate --name Note --attributes
title:string,description:string,userId:integer
```

- Migrations (validation)

```
# create-user.js
"use strict";

module.exports = {
  async up(queryInterface, Sequelize) {
    await queryInterface.createTable("Users", {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER,
      },
      name: {
        allowNull: false,
        type: Sequelize.STRING,
      },
      age: {
        allowNull: false,
        type: Sequelize.INTEGER,
      },
      email: {
        allowNull: false,
        unique: true,
        type: Sequelize.STRING,
      },
      password: {
        allowNull: false,
        type: Sequelize.STRING,
      },
      createdAt: {
```

```
        allowNull: false,
        type: Sequelize.DATE,
      },
      updatedAt: {
        allowNull: false,
        type: Sequelize.DATE,
      },
    });
  },
  async down(queryInterface, Sequelize) {
    await queryInterface.dropTable("Users");
  },
};

# create-note.js (references)
"use strict";

module.exports = {
  async up(queryInterface, Sequelize) {
    await queryInterface.createTable("Notes", {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER,
      },
      title: {
        allowNull: false,
        type: Sequelize.STRING,
      },
      description: {
        type: Sequelize.STRING,
      },
      userId: {
        allowNull: false,
        type: Sequelize.INTEGER,
```

```
        references: {
          model: "Users",
          key: "id",
        },
      },
      createdAt: {
        allowNull: false,
        type: Sequelize.DATE,
      },
      updatedAt: {
        allowNull: false,
        type: Sequelize.DATE,
      },
    });
  },
  async down(queryInterface, Sequelize) {
    await queryInterface.dropTable("Notes");
  },
};
```

- Models

```
# note.js
"use strict";
const { Model } = require("sequelize");

module.exports = (sequelize, DataTypes) => {
  class Note extends Model {
    static associate(models) {
      Note.belongsTo(models.User, {
        foreignKey: "userId",
      });
    }
  }
  Note.init(
```

```
    {
      title: {
        type: DataTypes.STRING,
        allowNull: false,
        validate: {
          notNull: { msg: "Title cannot be null" },
          notEmpty: { msg: "Title cannot be empty" },
        },
      },
      description: DataTypes.STRING,
      userId: {
        type: DataTypes.INTEGER,
        allowNull: false,
        validate: {
          notNull: { msg: "UserId cannot be null" },
          notEmpty: { msg: "UserId cannot be empty" },
        },
        references: {
          model: "Users",
          key: "id",
        },
      },
    },
    {
      sequelize,
      modelName: "Note",
    }
  );
  return Note;
};

# user.js
"use strict";
const { hashPassword } = require("../helpers/bcrypt");
const { Model } = require("sequelize");
```

```
module.exports = (sequelize, DataTypes) => {
  class User extends Model {
    static associate(models) {
      User.hasMany(models.Note, {
        foreignKey: "userId",
      });
    }
  }
  User.init(
    {
      name: {
        type: DataTypes.STRING,
        allowNull: false,
        validate: {
          notNull: { msg: "Name cannot be null" },
          notEmpty: { msg: "Name cannot be empty" },
        },
      },
      age: {
        type: DataTypes.INTEGER,
        allowNull: false,
        validate: {
          notNull: { msg: "Age cannot be null" },
          notEmpty: { msg: "Age cannot be empty" },
        },
      },
      email: {
        type: DataTypes.STRING,
        unique: true,
        allowNull: false,
        validate: {
          isEmail: { msg: "Email format is not correct" },
          notNull: { msg: "Email cannot be null" },
          notEmpty: { msg: "Email cannot be empty" },
        },
      },
```

```
      password: {
        type: DataTypes.STRING,
        allowNull: false,
        validate: {
          len: {
            args: [6],
            msg: "Password at least 6 characters",
          },
          notNull: { msg: "Password cannot be null" },
          notEmpty: { msg: "Password cannot be empty" },
        },
      },
    },
    {
      sequelize,
      modelName: "User",
    }
  );
  User.beforeCreate((user) => {
    user.password = hashPassword(user.password);
  });
  return User;
};
```

- Models (migrate the one without the Foreign Key first)

```
npx sequelize-cli db:migrate
```

- Seeders (add the command to sequelize.txt)

```
npx sequelize-cli seed:generate --name user
npx sequelize-cli seed:generate --name note
```

- Seeders

```
# user.js
"use strict";
const { hashPassword } = require("../helpers/bcrypt");

module.exports = {
  async up(queryInterface, Sequelize) {
    const data = require("../data/users.json");
    data.forEach((element) => {
      element.createdAt = new Date();
      element.updatedAt = new Date();
      element.password = hashPassword(element.password);
    });
    await queryInterface.bulkInsert("Users", data);
  },

  async down(queryInterface, Sequelize) {
    await queryInterface.bulkDelete("Users", null, {});
  },
};

# note.js
"use strict";

module.exports = {
  async up(queryInterface, Sequelize) {
    const data = require("../data/notes.json");
    data.forEach((element) => {
      element.createdAt = new Date();
      element.updatedAt = new Date();
    });
    await queryInterface.bulkInsert("Notes", data);
  },

  async down(queryInterface, Sequelize) {
    await queryInterface.bulkDelete("Notes", null, {});
```

```
  },
};
```

- Data

```
mkdir data && cd $_ && touch {users,notes}.json && cd ..

# users.json
[
  {
    "name": "user1",
    "age": 20,
    "email": "user1@example.com",
    "password": "123456"
  },
  {
    "name": "user2",
    "age": 25,
    "email": "user2@example.com",
    "password": "qwerty"
  }
]

# notes.json
[
  {
    "title": "Important Meeting Notes",
    "description": "Meeting with the team on [DATE]. Discu
ssed project progress, deadlines, and next steps. Key take
aways: [list of key points]",
    "userId": 1
  },
  {
    "title": "Grocery List",
    "description": "* Milk\n* Bread\n* Eggs\n* Fruits and
```

```
vegetables",
    "userId": 1
  },
  {
    "title": "Project Ideas",
    "description": "Brainstorming new project ideas. Some
potential options include: [list of ideas]",
    "userId": 1
  },
  {
    "title": "Weekend Plans",
    "description": "Planning activities for the upcoming w
eekend. Considering [list of options]",
    "userId": 2
  },
  {
    "title": "Movie Recommendations",
    "description": "A list of movies to watch: [list of mo
vie titles]",
    "userId": 2
  },
  {
    "title": "Book Wishlist",
    "description": "Books I'd like to read in the near fut
ure: [list of book titles]",
    "userId": 2
  }
]
```

- Helpers

```
mkdir helpers && cd $_ && touch {bcrypt,jwt}.js && cd ..

# bcrypt.js
const bcrypt = require("bcrypt");
```

```
function hashPassword(pass) {
  return bcrypt.hashSync(pass, bcrypt.genSaltSync(5));
}
function comparePassword(pass, hashedPass) {
  return bcrypt.compareSync(pass, hashedPass);
}

module.exports = { hashPassword, comparePassword };

# jwt.js
const jwt = require("jsonwebtoken");
const secret = process.env.JWT_SECRET;

function signToken(payload) {
  return jwt.sign(payload, secret);
}
function verifyToken(token) {
  return jwt.verify(token, secret);
}
module.exports = { signToken, verifyToken };
```

- Env

```
touch .env && touch .env.example && echo "node_modules" >
.gitignore && echo ".env" >> .gitignore

# .env
JWT_SECRET=secret

# .env.example
JWT_SECRET=string
```

- Seed

```
npx sequelize-cli db:seed:all
```

- Git (done)

```
git add . && git commit -am "migrate, model and seed done"
```

Express and Sequelize

- Index.js

```
# index.js
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

- Test server

```
# terminal
npm run start

# access with local browser
http://localhost:3000
```

- Index.js

```
# index.js
require("dotenv").config();
const express = require("express");
const app = express();
const port = 3000;

const HomeController = require("./controllers/homeControll
er");
const AuthController = require("./controllers/authControll
er");
const NoteController = require("./controllers/noteControll
er");

const errHandler = require("./middlewares/errorHandler");
const authentication = require("./middlewares/authenticati
on");
const authorization = require("./middlewares/authorizatio
n");

app.use(express.urlencoded({ extended: false }));
app.use(express.json());

// public
app.get("/", HomeController.home);
app.post("/register", AuthController.register);
app.post("/login", AuthController.login);

// authentication
app.use(authentication);

app.get("/note", NoteController.getNote);
app.post("/note", NoteController.addNote);

// authentication + authorization
app.patch(
```

```
  "/edit-description/:id",
  authorization,
  NoteController.editDescription
);
app.put("/edit-note/:id", authorization, NoteController.ed
itNote);
app.delete("/delete-note/:id", authorization, NoteControll
er.deleteNote);

app.use(errHandler);

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`);
  console.log(`http://localhost:${port}`);
});
```

- Middlewares

```
mkdir middlewares && cd $_ && touch {authentication,author
ization,errorHandler}.js && cd ..

# authentication.js
const { verifyToken } = require("../helpers/jwt");
const { User } = require("../models");

async function authentication(req, res, next) {
  try {
    const { authorization } = req.headers;
    if (!authorization) throw { name: "Invalid token" };

    const [type, access_token] = authorization.split(" ");
    if (type !== "Bearer") throw { name: "Invalid token"
};

    const verify = verifyToken(access_token);
```

```
    if (!verify) throw { name: "Invalid token" };

    const data = await User.findByPk(verify.id);
    if (!data) throw { name: "Invalid token" };

    req.user = data;
    next();
  } catch (error) {
    next(error);
  }
}

module.exports = authentication;

# authorization.js
const { Note } = require("../models");

async function authorization(req, res, next) {
  try {
    const data = await Note.findByPk(req.params.id);
    if (!data) throw { name: "Note not found" };
    if (data.userId !== req.user.id) throw { name: "You're
not authorized" };
    next();
  } catch (error) {
    next(error);
  }
}

module.exports = authorization;

# errorHandler.js
async function errHandler(error, req, res, next) {
  switch (error.name) {
    case "SequelizeValidationError":
    case "SequelizeUniqueConstraintError":
```

```
      const errors = error.errors[0].message;
      res.status(400).json({ message: errors });
      break;
    case "JsonWebTokenError":
    case "Invalid token":
      res.status(401).json({ message: "Invalid token" });
      break;
    case "Name is required":
      res.status(400).json({ message: "Name is required"
});
      break;
    case "Age is required":
      res.status(400).json({ message: "Age is required"
});
      break;
    case "Email is required":
      res.status(400).json({ message: "Email is required"
});
      break;
    case "Password is required":
      res.status(400).json({ message: "Password is require
d" });
      break;
    case "Title is required":
      res.status(400).json({ message: "Title is required"
});
      break;
    case "Invalid email/password":
      res.status(401).json({ message: "Invalid email/passw
ord" });
      break;
    case "Email already exist":
      res.status(401).json({ message: "Email already exis
t" });
      break;
    case "You're not authorized":
```

```
      res.status(403).json({ message: "You're not authoriz
ed" });
      break;
    case "Note not found":
      res.status(404).json({ message: "Note not found" });
      break;
    default:
      res.status(500).json({ message: "Internal server err
or" });
      break;
  }
}

module.exports = errHandler;
```

- Controllers

```
mkdir controllers && cd $_ && touch {auth,home,note}Contro
ller.js && cd ..

# authController.js
const { User } = require("../models");
const { comparePassword } = require("../helpers/bcrypt");
const { signToken } = require("../helpers/jwt");

class AuthController {
  static async register(req, res, next) {
    try {
      const { name, age, email, password } = req.body;
      if (!name) throw { name: "Name is required" };
      if (!age) throw { name: "Age is required" };
      if (!email) throw { name: "Email is required" };
      if (!password) throw { name: "Password is required"
};
```

```
      const findEmail = await User.findOne({
        where: { email },
      });
      if (findEmail) throw { name: "Email already exist"
};

      const data = await User.create({
        name,
        age,
        email,
        password,
      });
      res.status(201).json({
        id: data.id,
        name: data.name,
        age: data.age,
        email: data.email,
      });
    } catch (error) {
      next(error);
    }
  }
  static async login(req, res, next) {
    try {
      const { email, password } = req.body;
      if (!email) throw { name: "Email is required" };
      if (!password) throw { name: "Password is required"
};

      const data = await User.findOne({ where: { email }
});
      if (!data) throw { name: "Invalid email/password" };

      const checkPassword = comparePassword(password, dat
a.password);
      if (!checkPassword) throw { name: "Invalid email/pas
```

```
sword" };

      const payload = { id: data.id };
      const access_token = signToken(payload);

      res.status(200).json({ access_token });
    } catch (error) {
      next(error);
    }
  }
}

module.exports = AuthController;

# homeController.js
class HomeController {
  static async home(req, res, next) {
    try {
      res.status(200).json({ message: "Home" });
    } catch (error) {
      next(error);
    }
  }
}

module.exports = HomeController;

# noteController.js
const { Note } = require("../models");

class NoteController {
  // authentication
  static async getNote(req, res, next) {
    try {
      const data = await Note.findAll({
        where: { userId: req.user.id },
```

```
      });
      if (!data) throw { name: "Note not found" };
      res.status(200).json(data);
    } catch (error) {
      next(error);
    }
  }
  static async addNote(req, res, next) {
    try {
      const { title, description } = req.body;
      if (!title) throw { name: "Title is required" };

      const userId = req.user.id;
      await Note.create({
        title,
        description,
        userId,
      });

      res.status(201).json({ message: "Note has been adde
d" });
    } catch (error) {
      next(error);
    }
  }

  // authentication + authorization
  static async editDescription(req, res, next) {
    try {
      const noteId = req.params.id;
      const { description } = req.body;

      const note = await Note.findByPk(noteId);
      if (!note) throw { name: "Note not found" };

      note.description = description;
```

```
      await note.save();

      res.status(200).json({ message: "Note has been updat
ed" });
    } catch (error) {
      next(error);
    }
  }
  static async editNote(req, res, next) {
    try {
      const noteId = req.params.id;
      const { title, description } = req.body;

      const note = await Note.findByPk(noteId);
      if (!note) throw { name: "Note not found" };
      if (!title) throw { name: "Title is required" };

      note.title = title;
      note.description = description;
      await note.save();

      res.status(200).json({ message: "Note has been updat
ed" });
    } catch (error) {
      next(error);
    }
  }
  static async deleteNote(req, res, next) {
    try {
      const noteId = req.params.id;

      const data = await Note.findByPk(noteId);
      if (!data) throw { name: "Note not found" };
      await data.destroy();

      res.status(200).json({ message: "Note has been delet
```

```
ed" });
    } catch (error) {
      next(error);
    }
  }
}

module.exports = NoteController;
```

```
git add . && git commit -am "development done"
```

```
npm run start
```

Test

- Jest

```
mkdir __test__ && cd $_ && touch auth.test.js home.test.js
note.test.js && cd .. && npx sequelize-cli db:create --env
test && npx sequelize-cli db:migrate --env test && touch s
erver.js
```

- Index.js

```
# index.js
require("dotenv").config();
const express = require("express");
const app = express();

const HomeController = require("./controllers/homeControll
er");
const AuthController = require("./controllers/authControll
er");
```

```
const NoteController = require("./controllers/noteControll
er");

const errHandler = require("./middlewares/errorHandler");
const authentication = require("./middlewares/authenticati
on");
const authorization = require("./middlewares/authorizatio
n");

app.use(express.urlencoded({ extended: false }));
app.use(express.json());

// public
app.get("/", HomeController.home);
app.post("/register", AuthController.register);
app.post("/login", AuthController.login);

// authentication
app.use(authentication);

app.get("/note", NoteController.getNote);
app.post("/note", NoteController.addNote);

// authentication + authorization
app.patch(
  "/edit-description/:id",
  authorization,
  NoteController.editDescription
);
app.put("/edit-note/:id", authorization, NoteController.ed
itNote);
app.delete("/delete-note/:id", authorization, NoteControll
er.deleteNote);

app.use(errHandler);
```

```
module.exports = app;
```

- Package.json

```
# package.json
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "start": "nodemon server.js",
    "test": "npx jest --runInBand --forceExit --detectOpen
Handles --verbose",
    "coverage": "npx jest --coverage --runInBand --forceEx
it --detectOpenHandles --verbose"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "jsonwebtoken": "^9.0.2",
    "pg": "^8.11.5",
    "sequelize": "^6.37.3"
  },
  "devDependencies": {
    "jest": "^29.7.0",
    "nodemon": "^3.1.0",
    "sequelize-cli": "^6.6.2",
    "supertest": "^7.0.0"
```

```
  }
}
```

- Server.js

```
# server.js
const app = require("./index");
const port = process.env.PORT || 3000;

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`);
  console.log(`http://localhost:${port}`);
});
```

- Test

```
# auth.test.js
const request = require("supertest");
const app = require("../index");
const { User } = require("../models");

describe("POST /register", () => {
  test("Success register user", async () => {
    const data = {
      name: "user1",
      age: 20,
      email: "user1@example.com",
      password: "123456",
    };
    const res = await request(app).post("/register").send
(data);
    expect(res.status).toBe(201);
    expect(res.body).toMatchObject({
      id: expect.any(Number),
```

```
      name: data.name,
      age: data.age,
      email: data.email,
    });
  });

  test("Name is empty", async () => {
    const data = {
      age: 20,
      email: "user1@example.com",
      password: "123456",
    };
    const res = await request(app).post("/register").send
(data);
    expect(res.status).toBe(400);
    expect(res.body).toHaveProperty("message", "Name is re
quired");
  });

  test("Age is empty", async () => {
    const data = {
      name: "user1",
      email: "user1@example.com",
      password: "123456",
    };
    const res = await request(app).post("/register").send
(data);
    expect(res.status).toBe(400);
    expect(res.body).toHaveProperty("message", "Age is req
uired");
  });

  test("Email is empty", async () => {
    const data = {
      name: "user1",
      age: 20,
```

```
      password: "123456",
    };
    const res = await request(app).post("/register").send
(data);
    expect(res.status).toBe(400);
    expect(res.body).toHaveProperty("message", "Email is r
equired");
  });

  test("Password is empty", async () => {
    const data = {
      name: "user1",
      age: 20,
      email: "user1@example.com",
    };
    const res = await request(app).post("/register").send
(data);
    expect(res.status).toBe(400);
    expect(res.body).toHaveProperty("message", "Password i
s required");
  });

  test("password at least 6 characters", async () => {
    const data = {
      name: "user2",
      age: 20,
      email: "user2@example.com",
      password: "123",
    };
    const res = await request(app).post("/register").send
(data);
    expect(res.status).toBe(400);
    expect(res.body).toHaveProperty(
      "message",
      "Password at least 6 characters"
    );
```

```
  });

  test("Email already exist", async () => {
    const data = {
      name: "user1",
      age: 20,
      email: "user1@example.com",
      password: "123456",
    };
    const res = await request(app).post("/register").send
(data);

    expect(res.status).toBe(401);
    expect(res.body).toHaveProperty("message", "Email alre
ady exist");
  });

  test("Not a valid Email", async () => {
    const data = {
      name: "user1",
      age: 20,
      email: "user1exampleCom",
      password: "123456",
    };
    const res = await request(app).post("/register").send
(data);
    expect(res.status).toBe(400);
    expect(res.body).toHaveProperty("message", "Email form
at is not correct");
  });
});

describe("POST /login", () => {
  test("Success login user", async () => {
    const data = {
      email: "user1@example.com",
```

```
      password: "123456",
    };
    const res = await request(app).post("/login").send(dat
a);
    expect(res.status).toBe(200);
    expect(res.body).toHaveProperty("access_token", expec
t.any(String));
  });

  test("Email is required", async () => {
    const data = {
      password: "123456",
    };
    const res = await request(app).post("/login").send(dat
a);
    expect(res.status).toBe(400);
    expect(res.body).toHaveProperty("message", "Email is r
equired");
  });

  test("Password is required", async () => {
    const data = {
      email: "user1@example.com",
    };
    const res = await request(app).post("/login").send(dat
a);
    expect(res.status).toBe(400);
    expect(res.body).toHaveProperty("message", "Password i
s required");
  });

  test("Invalid Email", async () => {
    const data = {
      email: "user1exampleCom",
      password: "123456",
    };
```

```
    const res = await request(app).post("/login").send(dat
a);
    expect(res.status).toBe(401);
    expect(res.body).toHaveProperty("message", "Invalid em
ail/password");
  });

  test("Invalid Password", async () => {
    const data = {
      email: "user1@example.com",
      password: "123",
    };
    const res = await request(app).post("/login").send(dat
a);
    expect(res.status).toBe(401);
    expect(res.body).toHaveProperty("message", "Invalid em
ail/password");
  });
});

afterAll(async () => {
  await User.destroy({ truncate: true, cascade: true, rest
artIdentity: true });
});
```

```
# home.test.js
const request = require("supertest");
const app = require("../index");

describe("GET /", () => {
  test("Home", async () => {
    const response = await request(app).get("/");
    expect(response.status).toBe(200);
    expect(response.body).toHaveProperty("message", "Home");
```

```
  });
});
```

```
# note.test.js
const request = require("supertest");
const app = require("../index");
const { User, Note } = require("../models");
const { signToken } = require("../helpers/jwt");

let token1;
let token2;
beforeAll(async () => {
  let user1 = await User.create({
    name: "user1",
    age: 20,
    email: "user1@example.com",
    password: "123456",
  });
  token1 = signToken({ id: user1.id });

  let user2 = await User.create({
    name: "user2",
    age: 25,
    email: "user2@example.com",
    password: "qwerty",
  });
  token2 = signToken({ id: user2.id });
});

describe("POST /note", () => {
  test("Add new note 1", async () => {
    const data = {
      title: "Important Meeting Notes",
      description:
        "Meeting with the team on [DATE]. Discussed projec
```

```
t progress, deadlines, and next steps. Key takeaways: [lis
t of key points]",
    };
    const res = await request(app)
      .post("/note")
      .set("Authorization", `Bearer ${token1}`)
      .send(data);
    expect(res.status).toBe(201);
    expect(res.body).toHaveProperty("message", "Note has b
een added");
  });

  test("Add new note 2", async () => {
    const data = {
      title: "Weekend Plans",
      description:
        "Planning activities for the upcoming weekend. Con
sidering [list of options]",
    };
    const res = await request(app)
      .post("/note")
      .set("Authorization", `Bearer ${token2}`)
      .send(data);
    expect(res.status).toBe(201);
    expect(res.body).toHaveProperty("message", "Note has b
een added");
  });

  test("Title is required", async () => {
    const data = {
      description:
        "Meeting with the team on [DATE]. Discussed projec
t progress, deadlines, and next steps. Key takeaways: [lis
t of key points]",
    };
    const res = await request(app)
```

```
      .post("/note")
      .set("Authorization", `Bearer ${token1}`)
      .send(data);
    expect(res.status).toBe(400);
    expect(res.body).toHaveProperty("message", "Title is r
equired");
  });

  test("Invalid token 1", async () => {
    const data = {
      title: "Important Meeting Notes",
      description:
        "Meeting with the team on [DATE]. Discussed projec
t progress, deadlines, and next steps. Key takeaways: [lis
t of key points]",
    };
    const res = await request(app).post("/note").send(dat
a);
    expect(res.status).toBe(401);
    expect(res.body).toHaveProperty("message", "Invalid to
ken");
  });

  test("Invalid token 2", async () => {
    const data = {
      title: "Important Meeting Notes",
      description:
        "Meeting with the team on [DATE]. Discussed projec
t progress, deadlines, and next steps. Key takeaways: [lis
t of key points]",
    };
    const res = await request(app)
      .post("/note")
      .set("Authorization", `Bea asd`)
      .send(data);
    expect(res.status).toBe(401);
```

```
    expect(res.body).toHaveProperty("message", "Invalid to
ken");
  });
});

describe("GET /note", () => {
  test("Get note", async () => {
    const data = {
      title: "Important Meeting Notes",
      description:
        "Meeting with the team on [DATE]. Discussed projec
t progress, deadlines, and next steps. Key takeaways: [lis
t of key points]",
    };
    const res = await request(app)
      .get("/note")
      .set("Authorization", `Bearer ${token1}`);
    expect(res.status).toBe(200);
    expect(res.body).toMatchObject([
      {
        id: expect.any(Number),
        title: data.title,
        description: data.description,
        userId: expect.any(Number),
        createdAt: expect.any(String),
        updatedAt: expect.any(String),
      },
    ]);
  });

  test("Invalid token 1", async () => {
    const res = await request(app).get("/note");
    expect(res.status).toBe(401);
    expect(res.body).toHaveProperty("message", "Invalid to
ken");
  });
```

```
  test("Invalid token 2", async () => {
    const res = await request(app).get("/note").set("Autho
rization", `Bea asd`);
    expect(res.status).toBe(401);
    expect(res.body).toHaveProperty("message", "Invalid to
ken");
  });
});

describe("PATCH /edit-description/:id", () => {
  test(" note", async () => {
    const data = {
      description: "Patched description",
    };
    const res = await request(app)
      .patch(`/edit-description/${1}`)
      .set("Authorization", `Bearer ${token1}`)
      .send(data);
    expect(res.status).toBe(200);
    expect(res.body).toHaveProperty("message", "Note has b
een updated");
  });

  test("Invalid Token 1", async () => {
    const data = {
      description: "Patched description",
    };
    const res = await request(app).patch(`/edit-descriptio
n/${1}`).send(data);
    expect(res.status).toBe(401);
    expect(res.body).toHaveProperty("message", "Invalid to
ken");
  });

  test("Invalid Token 2", async () => {
```

```
    const data = {
      description: "Patched description",
    };
    const res = await request(app)
      .patch(`/edit-description/${1}`)
      .set("Authorization", `Bearer asd`)
      .send(data);
    expect(res.status).toBe(401);
    expect(res.body).toHaveProperty("message", "Invalid to
ken");
  });

  test("You're not authorized", async () => {
    const data = {
      description: "Patched description",
    };
    const res = await request(app)
      .patch(`/edit-description/${2}`)
      .set("Authorization", `Bearer ${token1}`)
      .send(data);
    expect(res.status).toBe(403);
    expect(res.body).toHaveProperty("message", "You're not
authorized");
  });

  test("Note not found", async () => {
    const data = {
      description: "Patched description",
    };
    const res = await request(app)
      .put(`/edit-note/${100}`)
      .set("Authorization", `Bearer ${token1}`)
      .send(data);
    expect(res.status).toBe(404);
    expect(res.body).toHaveProperty("message", "Note not f
ound");
```

```
  });
});

describe("PUT /edit-note/:id", () => {
  test("Edit note", async () => {
    const data = {
      title: "Updated title",
      description: "Updated description",
    };
    const res = await request(app)
      .put(`/edit-note/${1}`)
      .set("Authorization", `Bearer ${token1}`)
      .send(data);
    expect(res.status).toBe(200);
    expect(res.body).toHaveProperty("message", "Note has b
een updated");
  });

  test("Title is required", async () => {
    const data = {
      description: "Updated description",
    };
    const res = await request(app)
      .put(`/edit-note/${1}`)
      .set("Authorization", `Bearer ${token1}`)
      .send(data);
    expect(res.status).toBe(400);
    expect(res.body).toHaveProperty("message", "Title is r
equired");
  });

  test("Invalid Token 1", async () => {
    const data = {
      title: "Updated title",
      description: "Updated description",
    };
```

```
    const res = await request(app).put(`/edit-note/${1}`).
send(data);
    expect(res.status).toBe(401);
    expect(res.body).toHaveProperty("message", "Invalid to
ken");
  });

  test("Invalid Token 2", async () => {
    const data = {
      title: "Updated title",
      description: "Updated description",
    };
    const res = await request(app)
      .put(`/edit-note/${1}`)
      .set("Authorization", `Bearer asd`)
      .send(data);
    expect(res.status).toBe(401);
    expect(res.body).toHaveProperty("message", "Invalid to
ken");
  });

  test("You're not authorized", async () => {
    const data = {
      title: "Updated title",
      description: "Updated description",
    };
    const res = await request(app)
      .put(`/edit-note/${2}`)
      .set("Authorization", `Bearer ${token1}`)
      .send(data);
    expect(res.status).toBe(403);
    expect(res.body).toHaveProperty("message", "You're not
authorized");
  });

  test("Note not found", async () => {
```

```
    const data = {
      title: "Updated title",
      description: "Updated description",
    };
    const res = await request(app)
      .put(`/edit-note/${100}`)
      .set("Authorization", `Bearer ${token1}`)
      .send(data);
    expect(res.status).toBe(404);
    expect(res.body).toHaveProperty("message", "Note not f
ound");
  });
});

describe("DELETE /delete-note/:id", () => {
  test("Delete note", async () => {
    const res = await request(app)
      .delete(`/delete-note/${1}`)
      .set("Authorization", `Bearer ${token1}`);
    expect(res.status).toBe(200);
    expect(res.body).toHaveProperty("message", "Note has b
een deleted");
  });

  test("Invalid Token 1", async () => {
    const res = await request(app).delete(`/delete-note/
${2}`);
    expect(res.status).toBe(401);
    expect(res.body).toHaveProperty("message", "Invalid to
ken");
  });

  test("Invalid Token 2", async () => {
    const res = await request(app)
      .delete(`/delete-note/${2}`)
      .set("Authorization", `bea asd`);
```

```
    expect(res.status).toBe(401);
    expect(res.body).toHaveProperty("message", "Invalid to
ken");
  });

  test("You're not authorized", async () => {
    const res = await request(app)
      .delete(`/delete-note/${2}`)
      .set("Authorization", `Bearer ${token1}`);
    expect(res.status).toBe(403);
    expect(res.body).toHaveProperty("message", "You're not
authorized");
  });

  test("Note not found", async () => {
    const res = await request(app)
      .delete(`/delete-note/${100}`)
      .set("Authorization", `Bearer ${token1}`);
    expect(res.status).toBe(404);
    expect(res.body).toHaveProperty("message", "Note not f
ound");
  });
});

afterAll(async () => {
  await User.destroy({ truncate: true, cascade: true, rest
artIdentity: true });
  await Note.destroy({ truncate: true, cascade: true, rest
artIdentity: true });
});
```

- Test result

```
npm run test
```

- Test coverage

  ```
  npm run coverage
  ```

- Git (done)

  ```
  git add . && git commit -am "done"
  ```

Postman

https://documenter.getpostman.com/view/32679813/2sA3QnjaSb

````
touch README.md

# README.md
# Project: Express-Sequelize

## End-point: Home
### Method: GET
>```
>{{BaseUrl}}/
>```
### Response: 200
```json
{
    "message": "Home"
}
```



- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -


## End-point: Register
### Method: POST
````

````
>```
>{{BaseUrl}}/register
>```
### Body (**raw**)

```json
{
    "name": "user3",
    "age": 30,
    "email": "user@exampleCom",
    "password": "234567"
}
```

### Response: 201
```json
{
    "id": 3,
    "name": "user3",
    "age": 30,
    "email": "user3@example.com"
}
```

### Response: 400
```json
{
    "message": "Name is required"
}
```

### Response: 400
```json
{
    "message": "Age is required"
}
````
````

```

### Response: 400
```json
{
    "message": "Email is required"
}
```

Response: 400
```json
{
    "message": "Password is required"
}
```

Response: 400
```json
{
    "message": "Password at least 6 characters"
}
```

Response: 401
```json
{
    "message": "Email already exist"
}
```

Response: 400
```json
{
    "message": "Email format is not correct"
}
```
```

````
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -

## End-point: Login
### Method: POST
>```
>{{BaseUrl}}/login
>```
### Body (**raw**)

```json
{
    "email":"user1@example.com",
    "password":"123456"
}
```


### Response: 200
```json
{
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
pZCI6MSwiaWF0IjoxNzE2MzcxMzIyfQ.Uoru0UmmP2DMnxZ7dyr-0ycOrRwxQ
wSVUCKiX6CCduI"
}
```


### Response: 200
```json
{
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
pZCI6MiwiaWF0IjoxNzE2MzcxMzU4fQ.GeISJVZjxmtJTXvhurNh9Mm_tyqC2
qzJaQD2w_UQbic"
}
```
````

````
### Response: 400
```json
{
    "message": "Email is required"
}
```

### Response: 400
```json
{
    "message": "Password is required"
}
```

### Response: 401
```json
{
    "message": "Invalid email/password"
}
```

### Response: 401
```json
{
    "message": "Invalid email/password"
}
```



- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -

## End-point: Add Note
### Method: POST
>```
````

```
>{{BaseUrl}}/note
>```
Body (**raw**)

```json
{
    "title":"test_1",
    "description":"test_1"
}
```

🔑 Authentication bearer

Param	value	Type
token	{{access_token_2}}	string

Response: 201
```json
{
    "message": "Note has been added"
}
```

Response: 400
```json
{
    "message": "Title is required"
}
```

Response: 401
```json
{
    "message": "Invalid token"
```
```

````
}
```

### Response: 401
```json
{
    "message": "Invalid token"
}
```



- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## End-point: Get Note
### Method: GET
>```
>{{BaseUrl}}/note
>```
### 🔑 Authentication bearer

|Param|value|Type|
|---|---|---|
|token|{{access_token_1}}|string|



### Response: 200
```json
[
    {
        "id": 1,
        "title": "Important Meeting Notes",
        "description": "Meeting with the team on [DATE]. Discussed project progress, deadlines, and next steps. Key takeaways: [list of key points]",
        "userId": 1,
````

```
        "createdAt": "2024-05-22T09:47:59.408Z",
        "updatedAt": "2024-05-22T09:47:59.408Z"
    },
    {
        "id": 2,
        "title": "Grocery List",
        "description": "* Milk\n* Bread\n* Eggs\n* Fruits and
vegetables",
        "userId": 1,
        "createdAt": "2024-05-22T09:47:59.408Z",
        "updatedAt": "2024-05-22T09:47:59.408Z"
    },
    {
        "id": 3,
        "title": "Project Ideas",
        "description": "Brainstorming new project ideas. Some
potential options include: [list of ideas]",
        "userId": 1,
        "createdAt": "2024-05-22T09:47:59.408Z",
        "updatedAt": "2024-05-22T09:47:59.408Z"
    },
    {
        "id": 7,
        "title": "test_1",
        "description": "test_1",
        "userId": 1,
        "createdAt": "2024-05-22T09:52:01.629Z",
        "updatedAt": "2024-05-22T09:52:01.629Z"
    }
]
```

Response: 200
```json
[
    {
```

````
        "id": 4,
        "title": "Weekend Plans",
        "description": "Planning activities for the upcoming
weekend. Considering [list of options]",
        "userId": 2,
        "createdAt": "2024-05-22T09:47:59.408Z",
        "updatedAt": "2024-05-22T09:47:59.408Z"
    },
    {
        "id": 5,
        "title": "Movie Recommendations",
        "description": "A list of movies to watch: [list of m
ovie titles]",
        "userId": 2,
        "createdAt": "2024-05-22T09:47:59.408Z",
        "updatedAt": "2024-05-22T09:47:59.408Z"
    },
    {
        "id": 6,
        "title": "Book Wishlist",
        "description": "Books I'd like to read in the near fu
ture: [list of book titles]",
        "userId": 2,
        "createdAt": "2024-05-22T09:47:59.408Z",
        "updatedAt": "2024-05-22T09:47:59.408Z"
    }
]
```

### Response: 401
```json
{
    "message": "Invalid token"
}
```
````
````

````
### Response: 401
```json
{
    "message": "Invalid token"
}
```



- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -

## End-point: Edit Description
### Method: PATCH
>```
>{{BaseUrl}}/edit-description/:id
>```
### Body (**raw**)

```json
{
    "description":"test3"
}
```

### 🔑 Authentication bearer

|Param|value|Type|
|---|---|---|
|token|{{access_token_1}}|string|



### Response: 200
```json
{
    "message": "Note has been updated"
}
````
````

````
```

### Response: 401
```json
{
    "message": "Invalid token"
}
```

### Response: 401
```json
{
    "message": "Invalid token"
}
```

### Response: 403
```json
{
    "message": "You're not authorized"
}
```

### Response: 404
```json
{
    "message": "Note not found"
}
```



- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -

## End-point: Edit Note
### Method: PUT
````
````

````
>```
>{{BaseUrl}}/edit-note/:id
>```
### Body (**raw**)

```json
{
    "description":"test4"
}
```

### 🔑 Authentication bearer

|Param|value|Type|
|---|---|---|
|token|{{access_token_1}}|string|


### Response: 200
```json
{
    "message": "Note has been updated"
}
```

### Response: 400
```json
{
    "message": "Title is required"
}
```

### Response: 401
```json
{
    "message": "Invalid token"
````
````

````
}
```

### Response: 401
```json
{
    "message": "Invalid token"
}
```

### Response: 403
```json
{
    "message": "You're not authorized"
}
```

### Response: 404
```json
{
    "message": "Note not found"
}
```



- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -

## End-point: Delete Note
### Method: DELETE
>```
>{{BaseUrl}}/delete-note/:id
>```
### Body (**raw**)

```json
````

```

### 🔑 Authentication bearer

|Param|value|Type|
|---|---|---|
|token|{{access_token_1}}|string|


### Response: 200
```json
{
    "message": "Note has been deleted"
}
```

Response: 401
```json
{
    "message": "Invalid token"
}
```

Response: 401
```json
{
    "message": "Invalid token"
}
```

Response: 403
```json
{
    "message": "You're not authorized"
}
```
```

````
```


### Response: 404
```json
{
    "message": "Note not found"
}
```



- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
_________________________________________________
Powered By: [postman-to-markdown](https://github.com/bautistaj/postman-to-markdown/)
````

```
git add . && git commit -am "postman done"
```

Optional

Testing 100 % (incompatible with development)
```