САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнил:

Саунин Антон

Группа K33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

Нужно написать свой boilerplate на express + sequelize + typescript.

Должно быть явное разделение на:

- модели
- контроллеры
- роуты
- сервисы для работы с моделями
-

Ход работы

Написал файл для запуска сервера.

```
require("dotenv").config();

export default class App {
  public port: number;
  public host: string;

  private app: express.Application;
  private server: Server;

  constructor(port = 8000, host = "localhost") {
    this.port = Number(process.env.PORT) || port;
    this.host = process.env.HOST || host;

    this.app = this.createApp();
    this.server = this.createServer();
  }

  private createApp(): express.Application {
    const app = express();
    app.use(cors());
    app.use(bodyParser.json());
    app.use("/", routes);
    return app;
  }

  private createServer(): Server {
    const server = createServer(this.app);
    return server;
  }

  public start(): void {
    this.server.listen(this.port, () => {
      console.log(`Running server on port ${this.port}`);
    });
  }
}
```

Настроил Sequilize.

```
const sequelize = new Sequelize({
  database: "some_db",
  dialect: "sqlite",
  username: "root",
  password: "",
  storage: "db.sqlite",
  logging: console.log,
});

const models = [User, RefreshToken];

sequelize.addModels(models);

sequelize
  .sync()
  .then(() => {
    console.log("synced models");
  })
  .catch((e) => console.log(e));

async function testConnection() {
  try {
    await sequelize.authenticate();
    console.log("Connection has been established successfully.");
  } catch (error) {
    console.error("Unable to connect to the database:", error);
  }
}

testConnection();

export default sequelize;
```

Создано все для работы с пользователями.

Модель:

```ts
import hashPassword from '../../utils/hashPassword'
import { Table, Column, Model, Unique, AllowNull, BeforeCreate, BeforeUpda
import { Optional } from "sequelize";

export type UserAttributes = {
    id: string,
    name: string,
    email: string;
    password: string;
};

export type UserCreationAttributes = Optional<UserAttributes, 'id'>;


@Table
export class User extends Model<UserAttributes, UserCreationAttributes> {
    @PrimaryKey
    @Column({
        type: DataType.UUID,
        defaultValue: DataType.UUIDV4,
    })
    id: string;

    @Column
    name: string;

    @Unique
    @Column
    email: string

    @AllowNull(false)
    @Column
    password: string

    @BeforeCreate
    @BeforeUpdate
    static generatePasswordHash(instance: User) {
        const { password } = instance
        if (instance.changed('password')) {
            instance.password = hashPassword(password)
        }
    }
}

export default User
```

Контроллер:

```ts
export default class UserController {
    private userService: UserService

    constructor() {
        this.userService = new UserService()
    }

    get = async (request: any, response: any) => {
        try {
            const user = await this.userService.getAll()
            response.status(201).send(user)
        } catch (error: any) {
            response.status(404).send({ "error": error.toString() })
        }
    }

    update = async (request: any, response: any) => {
        try {
            const user = await this.userService.update(request.user, request.body)
            response.status(201).send(user)
        } catch (error: any) {
            response.status(400).send({ "error": error.toString() })
        }
    }

    delete = async (request: any, response: any) => {
        try {
            await this.userService.delete(request.user)
            response.status(201).send({ "error": 'User have successful deleted' })
        } catch (error: any) {
            response.status(400).send({ "error": error.toString() })
        }
    }

    me = async (request: any, response: any) => {
        response.send(request.user)
    }

    changePassword = async (request: any, response: any) => {
        try {
            await this.userService.changePassword(request.user, request.body)
            response.status(201).send({ "error": 'Password have successful changed' })
        } catch (error: any) {
            response.status(400).send({ "error": error.toString() })
        }
    }
}
```

Сервис:

```
class UserService {
    async getById(id: number): Promise<User> {
        try {
            const user = await User.findByPk(id)
            if (!user) {
                throw new Error('Not Found');
            }
            return user
        }
        catch (error) {
            throw error;
        }
    }

    async getAll(): Promise<User[]> {
        try {
            const user = await User.findAll()
            return user
        } catch (error) {
            throw error;
        }
    }

    async update(user: UserAttributes, userData: Pick<UserAttributes, 'email' | 'name'>): Promise<User> {
        try {
            const [updatedRowsCount, updatedUser] = await User.update(
                { name: userData.name, email: userData.email },
                {
                    where: { id: user.id },
                    returning: true,
                });

            if (updatedRowsCount === 0) {
                throw new Error('User not found');
            }
            return updatedUser[0];
        } catch (error) {
            throw error;
        }
    }
```

```
    async changePassword<Body extends {oldPassword: string, newPassword: string}>(user: UserAttributes, userData: Body): Promise<User> {
        try {
            if (!checkPassword(user, userData.oldPassword)) {
                throw new Error('Password is not correct');
            }
            const [updatedRowsCount, updatedUser] = await User.update(
                { password: hashPassword(userData.newPassword) },
                {
                    where: { id: user.id },
                    returning: true,
                });

            if (updatedRowsCount === 0) {
                throw new Error('User not found');
            }
            return updatedUser[0];
        } catch (error) {
            throw error;
        }
    }

    async delete(user: UserAttributes): Promise<number> {
        try {
            const deletedRowsCount = await User.destroy({ where: { id: user.id } });
            if (deletedRowsCount === 0) {
                throw new Error('User not found');
            }
            return deletedRowsCount;
        } catch (error) {
            throw error;
        }
    }
```

Эндпойнты:

```
const router: express.Router = express.Router();

const controller: UserController = new UserController();

router.get("/me", controller.me);
router
  .route("/")
  .get(controller.get)
  .patch(controller.update)
  .delete(controller.delete);
router.route("/password").patch(controller.changePassword);

export default router;
```

Далее был реализован механизм реализации:

Модель:

```ts
export type RefreshTokenType = {
    token: string;
    refreshToken: string;
}

@Table
export class RefreshToken extends Model {
    @Unique
    @AllowNull(false)
    @Column
    token: string

    @ForeignKey(() => User)
    @Column
    userId: string
}
```

Контроллер:

```ts
export default class AuthController {
    private authService: AuthService

    constructor() {
        this.authService = new AuthService()
    }

    login = async (request: any, response: any) => {
        try {
            const { email, password } = request.body;
            const data: RefreshTokenType | Error = await this.authService.login(email, password)
            response.status(201).send(data)
        } catch (error: any) {
            response.status(404).send({ "error": error.toString() })
        }
    }

    register = async (request: any, response: any) => {
        const { body } = request
        try {
            const data: UserAttributes | Error = await this.authService.register(body)
            response.status(201).send(data)
        } catch (error: any) {
            response.status(400).send({ "error": error.toString() })
        }
    }

    refreshToken = async (request: any, response: any) => {
        const { body } = request
        const { refreshToken } = body
        try {
            const data: RefreshTokenType | Error = await this.authService.refreshToken(refreshToken)
            response.status(201).send(data)
        } catch (error: any) {
            response.status(401).send({ "error": error.toString() })
        }
    }
}
```

Сервисы:

```ts
class RefreshTokenService {
    private user: User | null

    constructor(user: User | null = null) {
        this.user = user
    }

    generateRefreshToken = async (): Promise<string> => {
        const token = randomUUID()
        const userId = this.user?.id
        await RefreshToken.create({ token, userId })
        return token
    }

    isRefreshTokenExpired = async (token: string): Promise<{ userId: string | null, isExpired: boolean }> => {
        const refreshToken = await RefreshToken.findOne({ where: { token } })

        if (refreshToken) {
            const tokenData = refreshToken.toJSON()
            const currentDate = new Date()
            const timeDelta = currentDate.getTime() - tokenData.createdAt.getTime()
            if (timeDelta > 0 && timeDelta < 3600000) {
                return { userId: tokenData.userId, isExpired: false }
            }
        }

        return { userId: null, isExpired: true }
    }
}

export default RefreshTokenService
```

```
class AuthService {
    async register(userData: UserCreationAttributes): Promise<User> {
        try {
            const user = await User.create(userData)
            return user
        } catch (error) {
            throw error;
        }
    }

    async login(email: string, password: string): Promise<{ token: string, refreshToken: string }> {
        try {
            const user = await User.findOne({ where: { email } })
            if (!user || !checkPassword(user, password)) {
                throw new Error('Email or password is not correct');
            }
            const refreshTokenService = new RefreshTokenService(user)
            const refreshToken = await refreshTokenService.generateRefreshToken()
            const token = jwt.sign({ id: user.id }, SECRET_KEY, { expiresIn: '2 days' });
            return { refreshToken: refreshToken, token: token };
        } catch (error) {
            throw error;
        }
    }

    async refreshToken(refreshToken: string): Promise<{ token: string, refreshToken: string }> {
        const refreshTokenService = new RefreshTokenService()
        try {
            const { userId, isExpired } = await refreshTokenService.isRefreshTokenExpired(refreshToken)
            if (!isExpired && userId) {
                const user = await User.findByPk(userId)
                const refreshTokenService = new RefreshTokenService(user)
                const refreshToken = await refreshTokenService.generateRefreshToken()
                const accessToken = jwt.sign({ id: userId }, SECRET_KEY, { expiresIn: '2 days' });
                return { refreshToken: refreshToken, token: accessToken };
            } else {
                throw new Error('Invalid credentials')
            }
        } catch (error) {
            throw error;
        }
    }
}


export default AuthService
```

Эндпойнты:

```
const router: express.Router = express.Router();

const controller: AuthController = new AuthController();

router.post("/login", controller.login);
router.post("/register", controller.register);

export default router;
```

Middleware:

```
export const auth = async (req: Request, res: Response, next: NextFunction) => {
  try {
    const token = req.header('Authorization')?.replace('Bearer ', '');

    if (!token) {
      throw new Error('Missing token');
    }

    const decoded = jwt.verify(token, SECRET_KEY) as JwtPayload;
    const userData = await userService.getById(decoded.id);

    if (!userData) {
      throw new Error('User not found');
    }

    (req as CustomRequest).user = userData;
    next();
  } catch (err) {
    res.status(401).send({ "error": 'Please authenticate' });
  }
};
```

Также были написаны функции для хеширования и сравнивания пароля:

Хэширование:

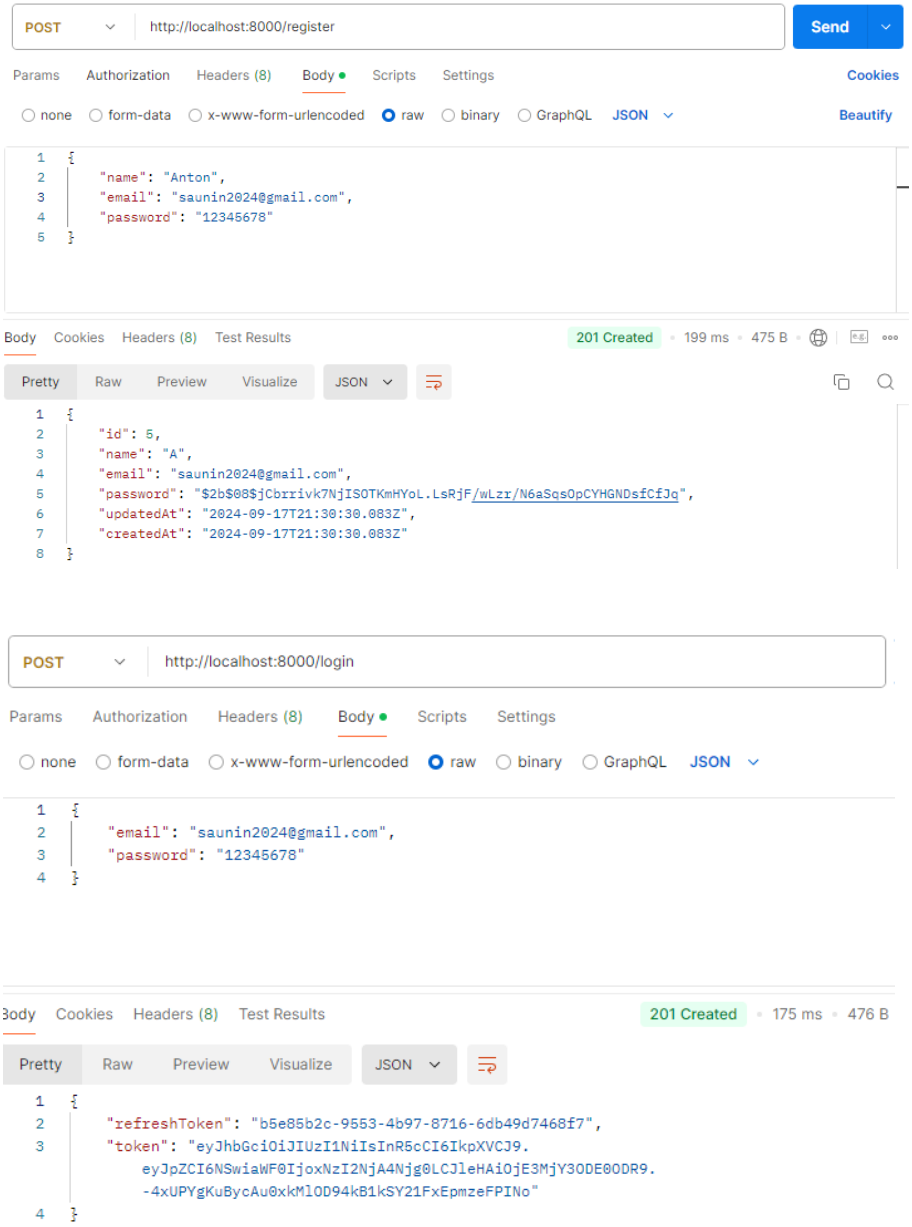
```
import argon2 from "argon2";

export default async (password: string): Promise<string> => {
  console.log(password);
  return await argon2.hash(password);
};
```

Проверка пароля:

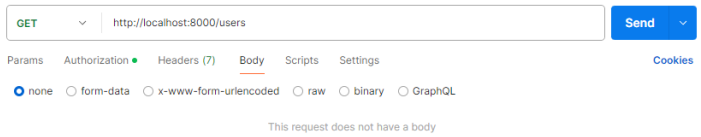
```
import argon2 from "argon2";
import { UserAttributes } from "../models/users/User";

export default async (
  user: UserAttributes,
  password: string
): Promise<boolean> => {
  return await argon2.verify(user.password, password);
};
```

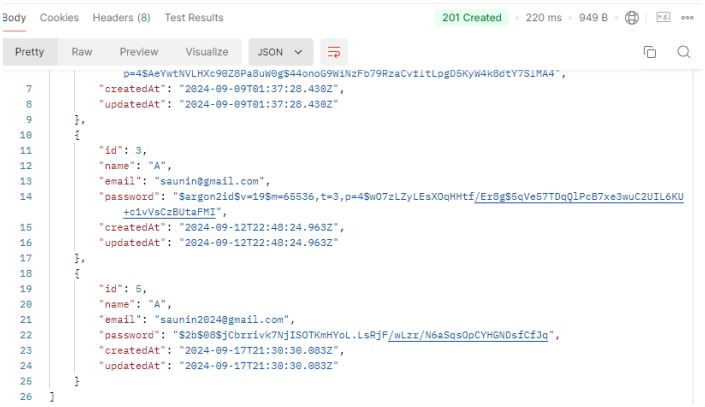
Проверим работу через postman:

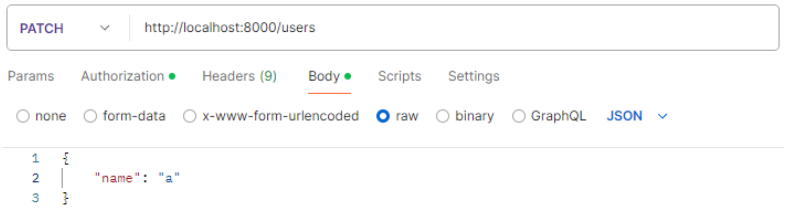
Авторизация:

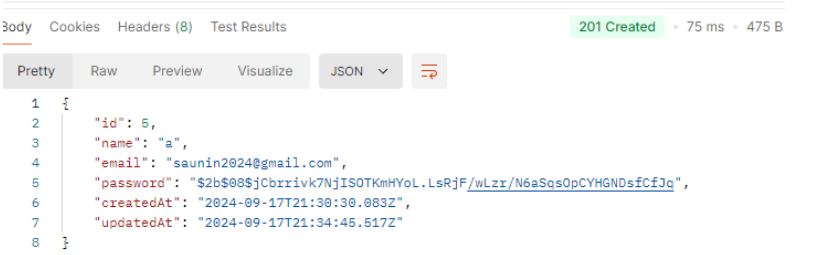


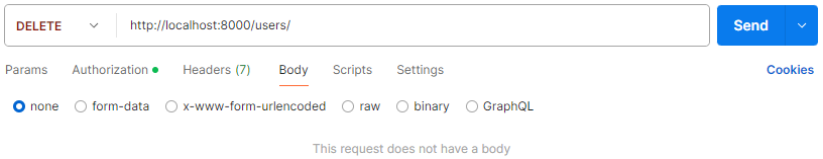
Users:

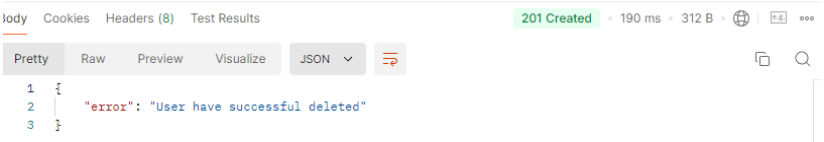

GET
http://localhost:8000/users
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
This request does not have a body


Body
Cookies
Headers (8)
Test Results
201 Created
220 ms
949 B
Pretty
Raw
Preview
Visualize
JSON
"createdAt": "2024-09-09T01:37:28.430Z",
"updatedAt": "2024-09-09T01:37:28.430Z"
"id": 3,
"name": "A",
"email": "saunin@gmail.com",
"createdAt": "2024-09-12T22:48:24.963Z",
"updatedAt": "2024-09-12T22:48:24.963Z"
"id": 5,
"name": "A",
"email": "saunin2024@gmail.com",
"password": "$2b$08$jCbrrivk7NjISOTKmHYoL.LsRjF/wLzr/N6aSqsOpCYHGNDsfCfJq",
"createdAt": "2024-09-17T21:30:30.083Z",
"updatedAt": "2024-09-17T21:30:30.083Z"


PATCH
http://localhost:8000/users
Params
Authorization
Headers (9)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"name": "a"
}


Body
Cookies
Headers (8)
Test Results
201 Created
75 ms
475 B
Pretty
Raw
Preview
Visualize
JSON
{
"id": 5,
"name": "a",
"email": "saunin2024@gmail.com",
"password": "$2b$08$jCbrrivk7NjISOTKmHYoL.LsRjF/wLzr/N6aSqsOpCYHGNDsfCfJq",
"createdAt": "2024-09-17T21:30:30.083Z",
"updatedAt": "2024-09-17T21:34:45.517Z"
}


DELETE
http://localhost:8000/users/
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
This request does not have a body


Body
Cookies
Headers (8)
Test Results
201 Created
190 ms
312 B
Pretty
Raw
Preview
Visualize
JSON
{
"error": "User have successful deleted"
}

Вывод

В ходе данной работы было создано API с авторизацией и возможности работы с пользователями