

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бек-энд разработка

Отчет по
лабораторной работе “Typescript: основы языка”

Выполнил:
Пронина Мария

Группа:
К33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задание:

Написать boilerplate на express + sequelize + typescript. Реализовать явное разделение на модели, контроллеры, роуты, сервисы для работы с моделями (реализовать паттерн “репозиторий”).

Ход работы:

1. Для реализации были установлены следующие зависимости:

express, sequelize-typescript, typescript, sqlite3, @types/express

2. Изменения в дефолтной конфигурации typescript: включение поддержки декораторов, директория сохранения js файлов и отключение конструктора для свойств класса:

```
"experimentalDecorators": true,  
  
"outDir": "./dist",  
  
"strictPropertyInitialization": false,
```

3. Создание базы данных:

```
const sequelize = new Sequelize({  
  database: 'lab1_db',  
  dialect: 'sqlite',  
  username: 'root',  
  password: '',  
  storage: 'db.sqlite',  
  logging: console.log,  
});
```

4. Создание модели:

```
@Table  
class Bear extends Model {  
  @PrimaryKey  
  @AutoIncrement  
  @Column(DataType.BIGINT)  
  id: number  
  
  @Column(DataType.TEXT)  
  name: string
```

```
@Column(DataType.FLOAT)
honey: number
}
```

5. Добавление модели в базу данных:

```
const models = [Bear]

sequelize.addModels(models)
```

6. Синхронизация моделей и подключение к базе данных

```
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)
  }
}
```

7. Сервис для получения объекта по id и создания нового объекта:

```
class BearService {
  async getById(id: number) : Promise<Bear> {
    try {
      const bear = await Bear.findByPk(id)
      if (!bear){
        throw new BearError(`Bear ${id} not found`);
      }
      return bear
    }
    catch (error){
      throw new BearError(`Bear error: ${error as Error}.message`)
    }
  }

  async create(bearData: any) : Promise<Bear> {
    try {
      const bear = await Bear.create(bearData)
      return bear
    }
  }
}
```

```

    }
    catch (error) {
        throw new BearError(`Bear error: ${ (error as
Error).message}`)
    }
}
}
}

```

8. Контроллер для работы с запросами и вызова методов сервиса:

```

class BearController {
    private bearService: BearService

    constructor() {
        this.bearService = new BearService()
    }

    get = async (req: Request, res: Response) => {
        try {
            const bear: Bear | BearError = await
this.bearService.getById(Number(req.params.id))
            res.status(200).send(bear)
        }
        catch (error) {
            res.status(404).send({"error": (error as Error).message})
        }
    }

    post = async (req: Request, res: Response) => {
        try {
            const bear : Bear | BearError = await
this.bearService.create(req.body)
            res.status(201).send(bear)
        }
        catch (error) {
            res.status(400).send({"error": (error as Error).message})
        }
    }
}

```

9. Реализация роутера:

```
import { Router } from 'express'
import BearController from '../controllers/bears/controller'

const router = Router()
const controller = new BearController()

router.get('/bear/:id', controller.get)
router.post('/bear', controller.post)
```

10. Создание приложения, подключение к порту:

```
const app = express();
app.use(express.json());
app.use('/', router);

const port = 3000
app.listen(port, () => {
  sequelize
  console.log(`listening on port ${port}`);
})
```

11.Выполнение запросов:

The screenshot shows a REST client interface. At the top, a POST request is configured to `http://127.0.0.1:3000/bear`. The 'Body' tab is selected, and the request body is a JSON object: `{ "name": "oleg", "honey": 100 }`. Below the request, the response is displayed in the 'Body' tab, showing a JSON object with the following fields: `{ "id": 7, "name": "oleg", "honey": 100, "updatedAt": "2024-04-03T20:56:38.309Z", "createdAt": "2024-04-03T20:56:38.309Z" }`. The status bar at the bottom indicates a 201 status code, 'Created' method, 206 ms response time, and 353 B response size.

```
POST http://127.0.0.1:3000/bear

{
  "name": "oleg",
  "honey": 100
}
```

```
{
  "id": 7,
  "name": "oleg",
  "honey": 100,
  "updatedAt": "2024-04-03T20:56:38.309Z",
  "createdAt": "2024-04-03T20:56:38.309Z"
}
```

201 Created 206 ms 353 B

GET

▼

http://127.0.0.1:3000/bear/7

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Query Params

	Key	Value
	Key	Value

Body

Cookies

Headers (7)

Test Results

200 OK

17 ms

348 B

Pretty

Raw

Preview

Visualize

JSON ▼

≡

```
1  {
2    "id": 7,
3    "name": "oleg",
4    "honey": 100,
5    "createdAt": "2024-04-03T20:56:38.309Z",
6    "updatedAt": "2024-04-03T20:56:38.309Z"
7  }
```

Вывод:

В ходе лабораторной работы был реализован boilerplate на typescript + express, создана структура, явно разделяющая проект на модели, контроллеры и сервисы.