САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет
Практическая работа №2
“Знакомство с fastify и prisma”

Выполнил:
Стукалов Артем
K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задание:

Реализовать базовое REST API с использованием fastify и prisma:

1) Продумать свою собственную модель пользователя.
2) Реализовать набор из CRUD-методов для работы с пользователями средствами Fastify + Prisma.
3) Написать запрос для получения пользователя по id/email.

P.S: первоначально в практической работе предполагалось использование express + sequelize. Но данные библиотеки были заменены на fastify + prisma по следующим причинам:

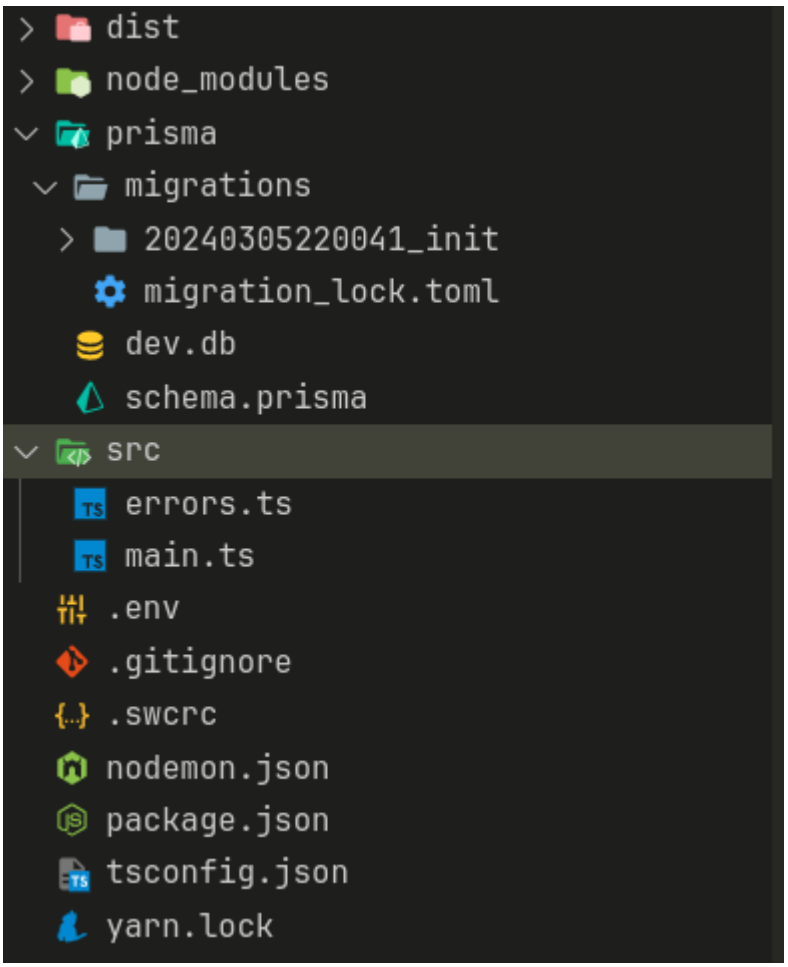
1) Fastify является более новой библиотекой относительно express и обладает заметно большей производительностью. Также она создавалась сразу с прицелом на использование с TS, что повышает ее типобезопасность.
2) Prisma имеет более удобный интерфейс формат написания схемы БД, которая впоследствии автоматически преобразуется в типы TypeScript, что опять же делает разработку более типобезопасной.

Конфигурация репозитория

В основе как уже было сказано ранее используется fastify + prisma. В качестве компилятора TypeScript используется SWC. Он написан на Rust, что сильно повышает скорость сборки проекта. Как при холодный старт на моей локальной машине происходит за 67 мс, а hot-reload пересборка за 3мс. Таким образом в связке с nodemon для перезапуска сервера мы ограничены только скоростью самой node-js.

В детали реализации работы SWC в связке с nodemon в рамках данного отчета не будем. Эта тема будет рассмотрена более подробно при реализации первой лабораторной работы.

Структура папок получилась следующая: папка src с исходниками приложения и автоматически сгенерированная пара prisma, которая содержит в себе файл самой базы данных, схему базы данных и миграции.




```json
{
  "name": "hw2",
  "version": "1.0.0",
  "description": "",
  ▷ Debug
  "scripts": {
    "dev": "concurrently --kill-others -n swc-build,node \"yarn build-watch\" \"yarn dev-watch\"",
    "dev-watch": "cross-env NODE_ENV=development nodemon --config nodemon.json",
    "build-watch": "swc ./src -w --strip-leading-paths --config-file ./.swcrc -d dist",
    "build": "swc ./src --strip-leading-paths --config-file ./.swcrc -d dist",
    "clean": "rimraf ./dist",
    "preview": "cross-env NODE_ENV=production node ./dist/main.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@prisma/client": "^5.10.2",
    "fastify": "^4.26.2"
  },
  "devDependencies": {
    "@swc/cli": "^0.3.10",
    "@swc/core": "^1.4.2",
    "@types/node": "^20.11.24",
    "chokidar": "^3.6.0",
    "concurrently": "^8.2.2",
    "cross-env": "^7.0.3",
    "nodemon": "^3.1.0",
    "prisma": "^5.10.2",
    "rimraf": "^5.0.5",
    "ts-node": "^10.9.2",
    "typescript": "^5.3.3"
  }
}
```

CRUD методы и модель пользователя:

Модель пользователя была сделана максимально простой и состоит из 4 полей: уникальный идентификатор, имя, почта и хеша пароля.

```
generator client {
  provider = "prisma-client-js"
}

You, in 2 hours | 1 author (You)
datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}

You, in 2 hours | 1 author (You)
model User {
  id       Int    @id @default(autoincrement())
  username String
  email    String @unique
  passHash String
}
```

В качестве примеров приведу минимальный рабочий файл main.ts с одним методом. Реализацию остальных методов можно посмотреть в исходниках репозитория.

```ts
import fastify from 'fastify'
import { PrismaClient } from '@prisma/client'
import { createHmac } from 'crypto'
import { AuthenticationError } from './errors'

const prisma = new PrismaClient()
const app = fastify({ logger: true })

// Просто для примера, не надо так делать
const TEMP_SECRET_KEY = 'NO_WAIFU_NO_WIFI'

app.get('/ping', async () => {
  return 'pong'
})

type SignupBody = {
  username: string
  email: string
  password: string
}

app.post<{
  Body: SignupBody
}>(`/signup`, async (req) => {
  const { username, email, password } = req.body

  const user = await prisma.user.create({
    data: {
      email,
      username,
      passHash: createHmac('sha256', TEMP_SECRET_KEY)
        .update(password)
        .digest('hex'),
    },
  })

  return user
})
```

```ts
type SigninBody = {
  username: string
  password: string
}

app.post<{
  Body: SigninBody
}>(`/signin`, async (req) => {
  const { username, password } = req.body

  const passHash = createHmac('sha256', TEMP_SECRET_KEY)
    .update(password)
    .digest('hex')

  const user = await prisma.user.findFirst({
    where: {
      username,
      passHash,
    },
  })

  if (!user) {
    throw new AuthenticationError()
  }

  return user
})

async function main() {
  try {
    await app.listen({ port: 3000 })
    console.log('🚀 Server ready at: http://localhost:3000')
  } catch (err) {
    console.error(err)
    prisma.$disconnect()
    process.exit(1)
  }
}
main()
```

Выводы:

Были получены навыки работы с библиотеками fastify и prisma.