САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа 2

Выполнил:

Беломытцев Андрей

К3339

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Реализовать все модели данных, спроектированные в рамках ДЗ1

Реализовать набор из CRUD-методов для работы с моделями данных средствами Express + TypeScript

Реализовать API-эндпоинт для получения пользователя по id/email

Ход работы

Файл app.ts реализован следующим образом

```
import express from 'express'
import { AppDataSource } from "./AppDataSource"
import categoryRouter from "./routers/Category"
import channelRouter from "./routers/Channel"
import reviewRouter from "./routers/Review"
import roleRouter from "./routers/Role"
import themeRouter from "./routers/Theme"
import userRouter from "./routers/User"
import videoRouter from "./routers/Video"

const app = express()
const PORT = 3000

app.use(express.json())
app.use('/category', categoryRouter)
app.use('/channel', channelRouter)
app.use('/review', reviewRouter)
app.use('/role', roleRouter)
app.use('/theme', themeRouter)
app.use('/user', userRouter)
app.use('/video', videoRouter)

AppDataSource.initialize().then(() => {
  app.listen(PORT, () => {
    console.log(`http://127.0.0.1:${PORT}`)
  })
}).catch(error => console.log(error))
```

Для подключения к базе данных прописаны следующие настройки в файле AppDataSource.ts

```
import { DataSource } from 'typeorm'
import { Category } from './models/Category'
import { Channel } from './models/Channel'
import { Review } from './models/Review'
import { Role } from './models/Role'
import { Theme } from './models/Theme'
import { User } from './models/User'
import { Video } from './models/Video'

export const AppDataSource = new DataSource({
  type: 'postgres',
```

```
  host: 'localhost',
  port: 5432,
  username: 'user',
  password: 'qwerty',
  database: 'db',
  synchronize: true,
  logging: true,
  entities: [
    Category,
    Channel,
    Review,
    Role,
    Theme,
    User,
    Video,
  ],
  migrations: [],
  subscribers: [],
})
```

А для удобного использования базы данных PostgreSQL применялся Docker.

```
docker run --name postgres \
  -e POSTGRES_USER=user \
  -e POSTGRES_PASSWORD=qwerty \
  -e POSTGRES_DB=db \
  -p 5432:5432 \
  -d postgres
```

На основе схемы базы данных были созданы все необходимые модели и размещены в папке models. Рассмотрим на примере Channel.

```
import { Entity, PrimaryColumn, Column, ManyToOne, OneToMany,
CreateDateColumn, UpdateDateColumn } from 'typeorm'
import { Review } from './Review'
import { Video } from './Video'
import { User } from './User'
import { Category } from './Category'
import { Theme } from './Theme'

@Entity()
export class Channel {
  @PrimaryColumn()
  id: string

  @Column()
  url: string

  @Column()
  title: string

  @Column('bigint')
  views: number

  @Column('int')
  subs: number

  @Column('int')
```

```
  videos: number

  @Column()
  lang: string

  @ManyToOne(type => Category, category => category.channels)
  category: Category

  @ManyToOne(type => Theme, theme => theme.channels)
  theme: Theme

  @Column()
  iconDefault: string

  @Column()
  iconMedium: string

  @Column()
  iconHigh: string

  @Column()
  description: string

  @Column()
  isApproved: boolean

  @ManyToOne(type => User, user => user.channels)
  user: User

  @CreateDateColumn()
  timeCreate: Date

  @UpdateDateColumn()
  timeUpdate: Date

  @OneToMany(type => Review, review => review.user)
  reviews: Review[]

  @OneToMany(type => Video, video => video.channel)
  videosList: Video[]
}
```

Другие модели аналогично.

Были реализованы CRUD-методы и размещены в папке controllers.

```
import { Request, Response } from 'express'
import { AppDataSource } from "../AppDataSource"
import { Channel } from '../models/Channel'

const repository = AppDataSource.getRepository(Channel)

const create = async (req: Request, res: Response) => {
  try {
    res.json(await repository.save(req.body))
  } catch (err: any) {
    res.status(500).json({ error: err.message })
  }
}
```

```
const get = async (req: Request, res: Response) => {
  try {
    res.json(await repository.find())
  } catch (err: any) {
    res.status(500).json({ error: err.message })
  }
}

const getOne = async (req: Request, res: Response) => {
  try {
    const id = req.params.id
    res.json(await repository.findOneBy({ id }))
  } catch (err: any) {
    res.status(500).json({ error: err.message })
  }
}

const update = async (req: Request, res: Response) => {
  try {
    const id = req.params.id
    const x = await repository.findOneBy({ id })
    if (!x) {
      res.status(404).json({ error: 'Not found' })
    }
    else {
      repository.merge(x, req.body)
      res.json(await repository.save(x))
    }
  } catch (err: any) {
    res.status(500).json({ error: err.message })
  }
}

const remove = async (req: Request, res: Response) => {
  try {
    const r = await repository.delete(req.params.id)
    if (r.affected === 0) {
      res.status(404).json({ error: 'Not found' })
    }
    else{
      res.json(r)
    }
  } catch (err: any) {
    res.status(500).json({ error: err.message })
  }
}

export default { create, get, getOne, update, remove }
```

Другие CRUD аналогично.

Были реализованы роутеры и размещены в папке routers.

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

const router = Router()

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

```
router.put('/:id', controller.update)
router.delete('/:id', controller.remove)

export default router
```

Другие роутеры аналогично.

Необходимо было также реализовать получение пользователя по id/email. По id уже реализовано аналогично с другими. А по email реализовано следующим образом.

```
const getByEmail = async (req: Request, res: Response) => {
  try {
    const email = req.params.email
    const user = await repository.findOne({ where: { email: email } })
    if (!user) {
      res.status(404).json({ message: 'Not found' })
    }
    else {
      res.json(user)
    }
  } catch (err: any) {
    res.status(500).json({ error: err.message })
  }
}
```

А в роутере

```
router.get('/email/:email', controller.getByEmail)
```

Вывод

В результате были реализованы все необходимые модели данных, которые были спроектированы в ДЗ1, с помощью TypeORM. Разработаны CRUD-методы для каждой модели с использованием Express и TypeScript. Добавлена возможность получения пользователя по id и email.