САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2 «Тестирование, разработка и документирование RESTful API»

Выполнила:
Коник А. А.
Группа K33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

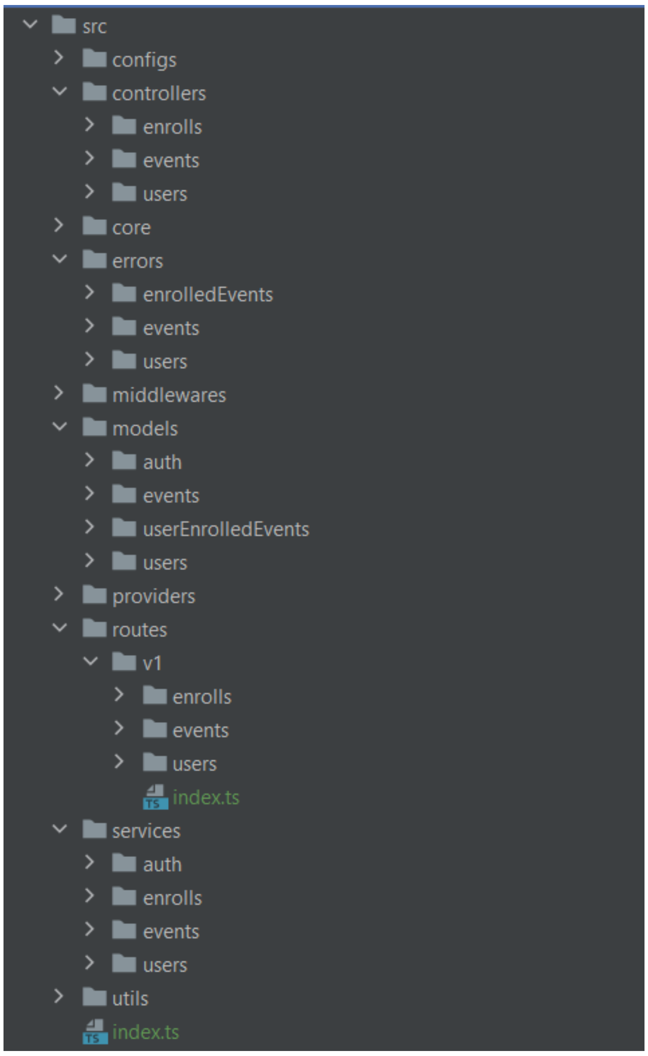
По выбранному варианту необходимо будет реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate).

Вариант:

Платформа для поиска мероприятий в Санкт-Петербурге.

Ход работы

Структура приложения:



```
src
  configs
  controllers
    enrolls
    events
    users
  core
  errors
    enrolledEvents
    events
    users
  middlewares
  models
    auth
    events
    userEnrolledEvents
    users
  providers
  routes
    v1
      enrolls
      events
      users
      index.ts
  services
    auth
    enrolls
    events
    users
  utils
  index.ts
```

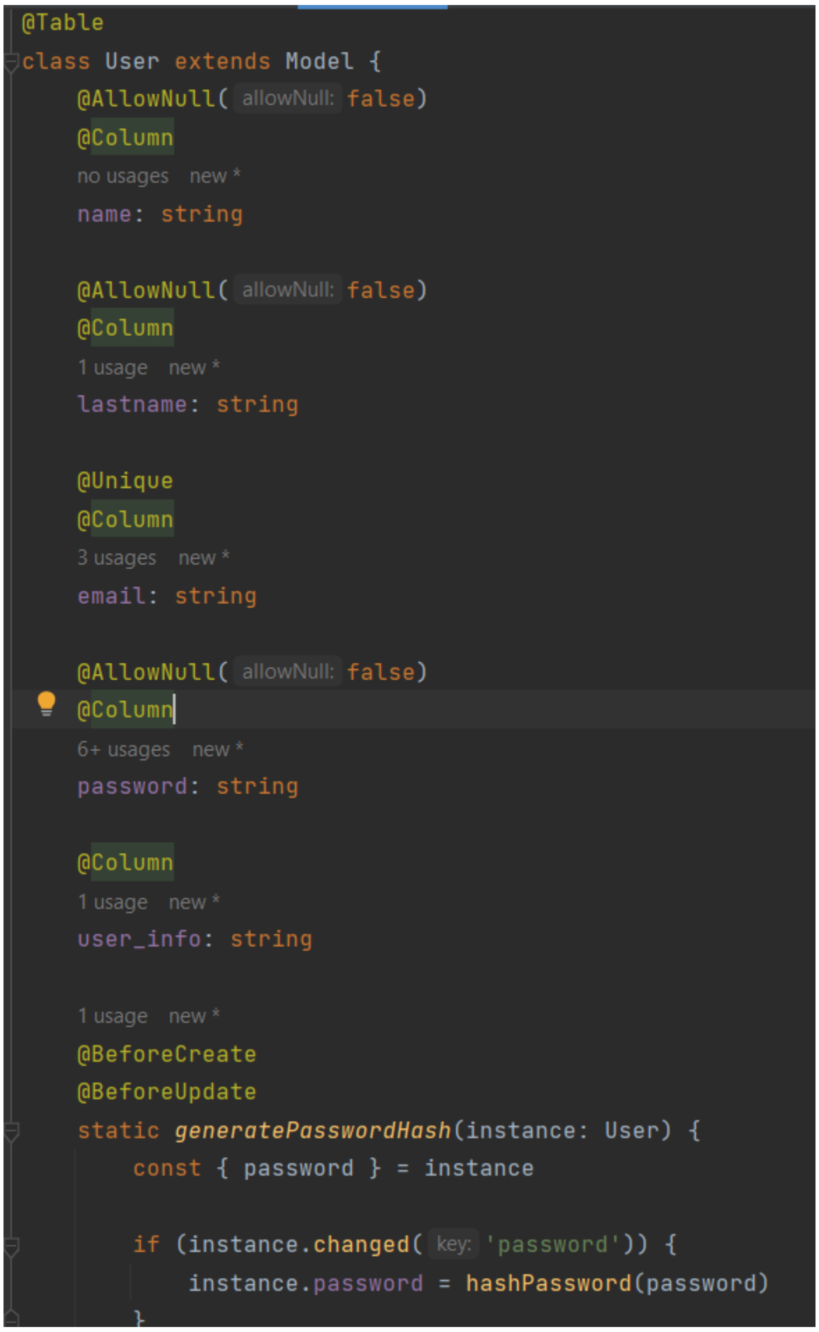
Где

- **core** – точка входа в приложение;
- **configs** – файлы конфигурации (файл для подключения к БД);
- **controllers** – контроллеры, отвечающие за логику обработки http-запросов;
- **models** – модели sequelize;
- **providers** – точки доступа к данным;

- **routes** – описание маршрутов;
- **services –** службы, которые содержат запросы к базе данных и возвращают объекты или выдают ошибки;
- **utils** – вспомогательные файлы, которые используются в приложении;
- **middlewares -** содержит аутентификацию с использованием passport.ts.

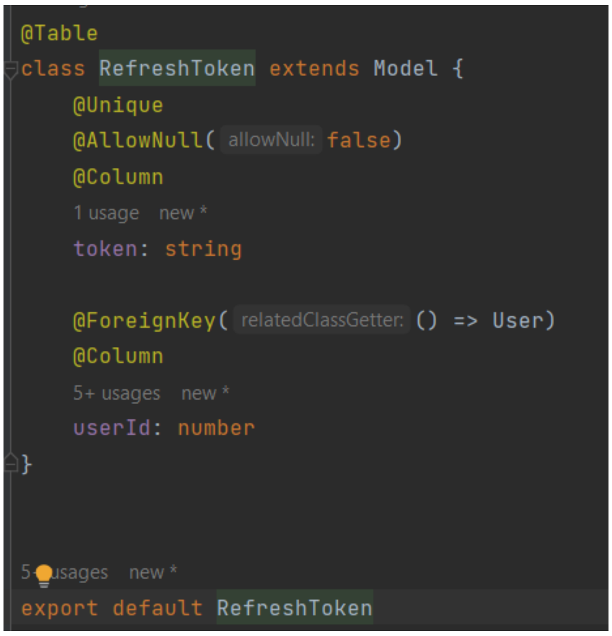
Модели:

User.ts – модель пользователя



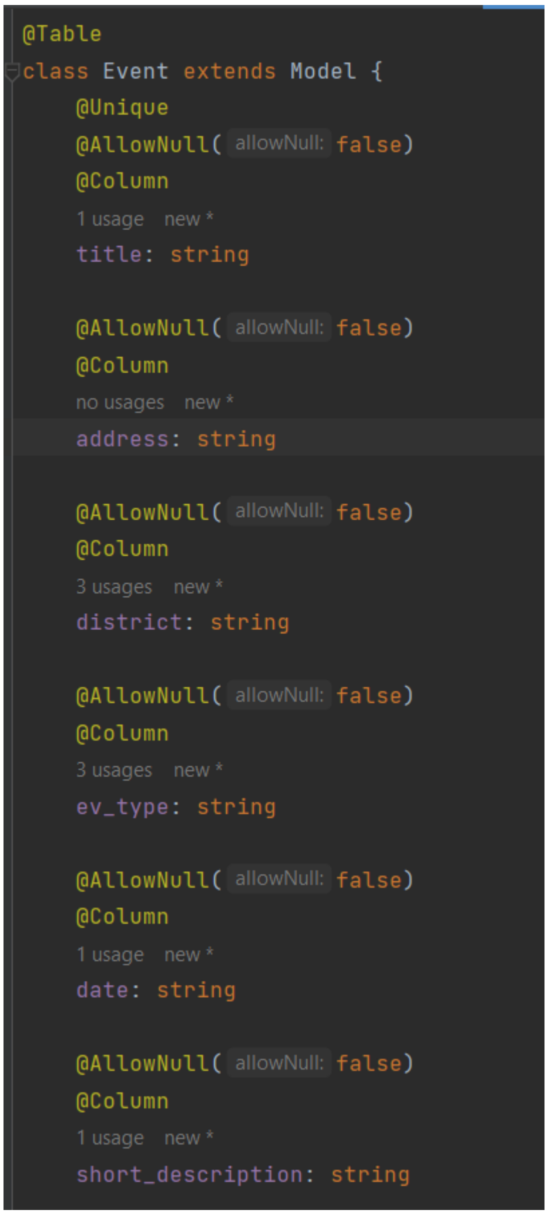
```
@Table
class User extends Model {
    @AllowNull( allowNull: false)
    @Column
    no usages   new *
    name: string

    @AllowNull( allowNull: false)
    @Column
    1 usage   new *
    lastname: string

    @Unique
    @Column
    3 usages   new *
    email: string

    @AllowNull( allowNull: false)
    @Column
    6+ usages   new *
    password: string

    @Column
    1 usage   new *
    user_info: string

    1 usage   new *
    @BeforeCreate
    @BeforeUpdate
    static generatePasswordHash(instance: User) {
        const { password } = instance

        if (instance.changed( key: 'password')) {
            instance.password = hashPassword(password)
        }
```

RefreshToken.ts - модель хранения токенов

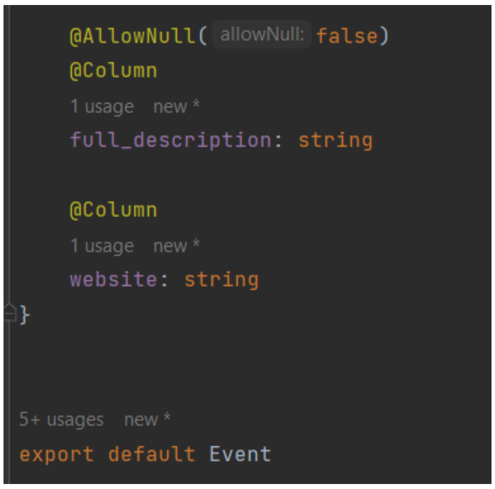


```
@Table
class RefreshToken extends Model {
    @Unique
    @AllowNull( allowNull: false)
    @Column
    1 usage   new *
    token: string

    @ForeignKey( relatedClassGetter: () => User)
    @Column
    5+ usages   new *
    userId: number
}


5 usages   new *
export default RefreshToken
```

Event.ts – модель мероприятия

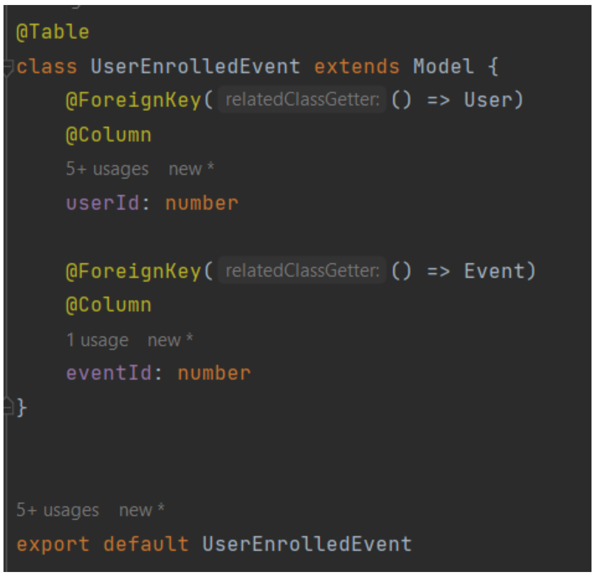


```
@Table
class Event extends Model {
    @Unique
    @AllowNull( allowNull: false)
    @Column
    1 usage   new *
    title: string

    @AllowNull( allowNull: false)
    @Column
    no usages   new *
    address: string

    @AllowNull( allowNull: false)
    @Column
    3 usages   new *
    district: string

    @AllowNull( allowNull: false)
    @Column
    3 usages   new *
    ev_type: string

    @AllowNull( allowNull: false)
    @Column
    1 usage   new *
    date: string

    @AllowNull( allowNull: false)
    @Column
    1 usage   new *
    short_description: string
```



```
    @AllowNull( allowNull: false)
    @Column
    1 usage   new *
    full_description: string

    @Column
    1 usage   new *
    website: string
}


5+ usages   new *
export default Event
```

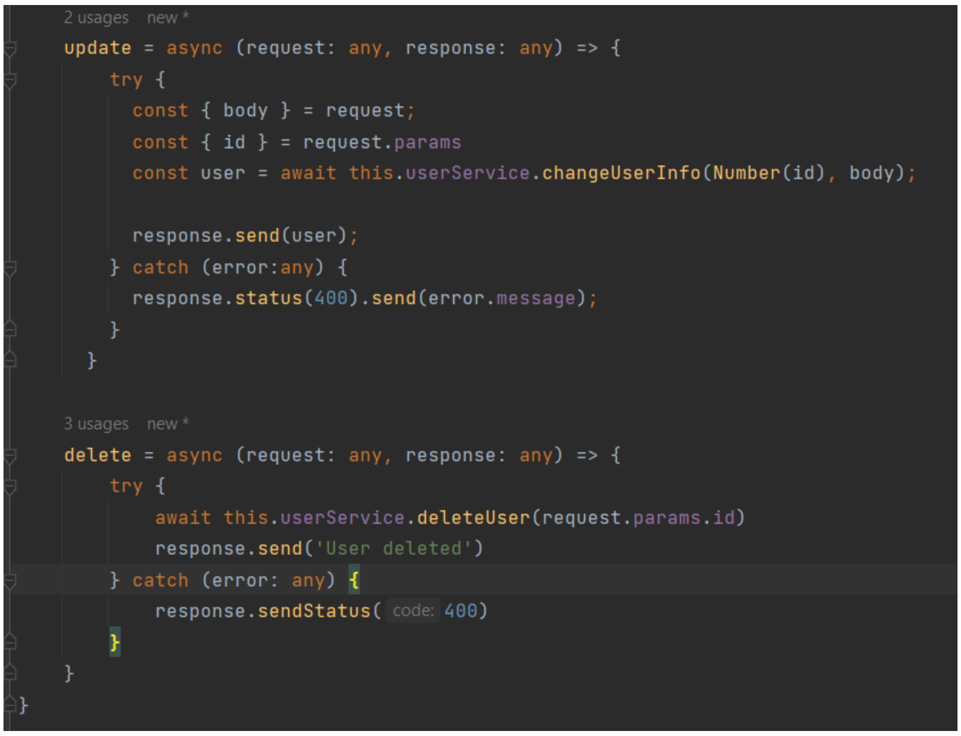
UserEnrolledEvents.ts – модель мероприятий пользователей



```
@Table
class UserEnrolledEvent extends Model {
    @ForeignKey( relatedClassGetter: () => User)
    @Column
    5+ usages   new *
    userId: number

    @ForeignKey( relatedClassGetter: () => Event)
    @Column
    1 usage   new *
    eventId: number
}


5+ usages   new *
export default UserEnrolledEvent
```

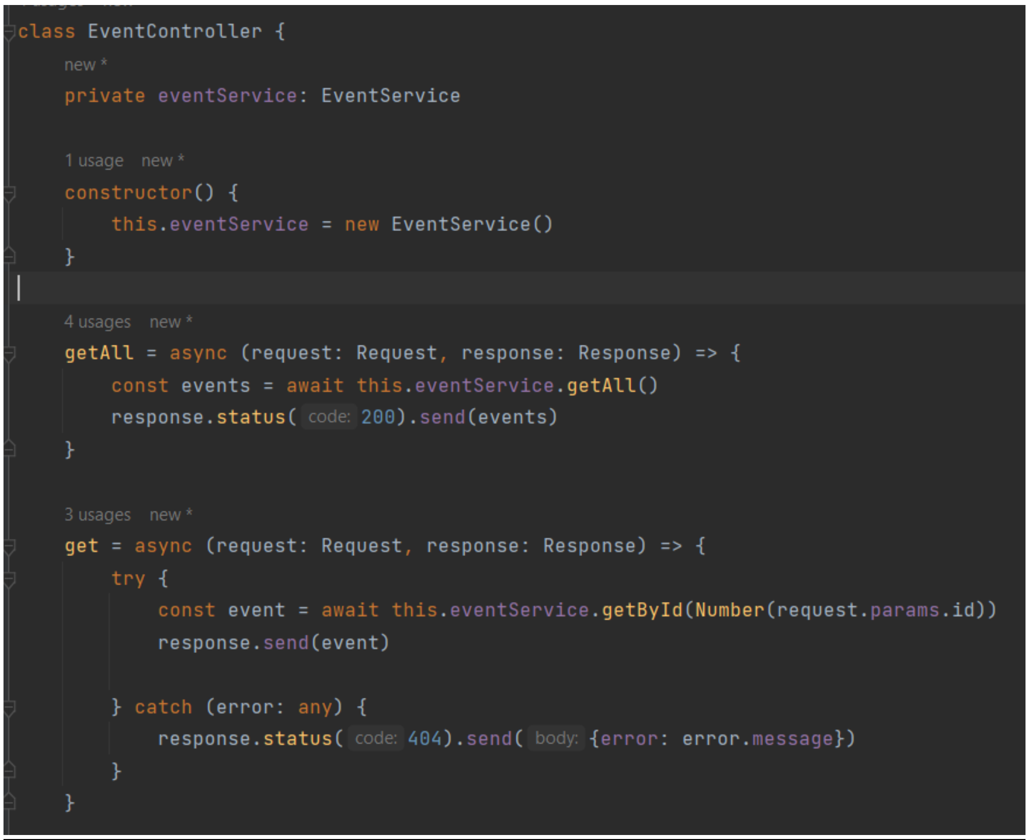
<u>Новые функции в контроллерах:</u>

User.ts

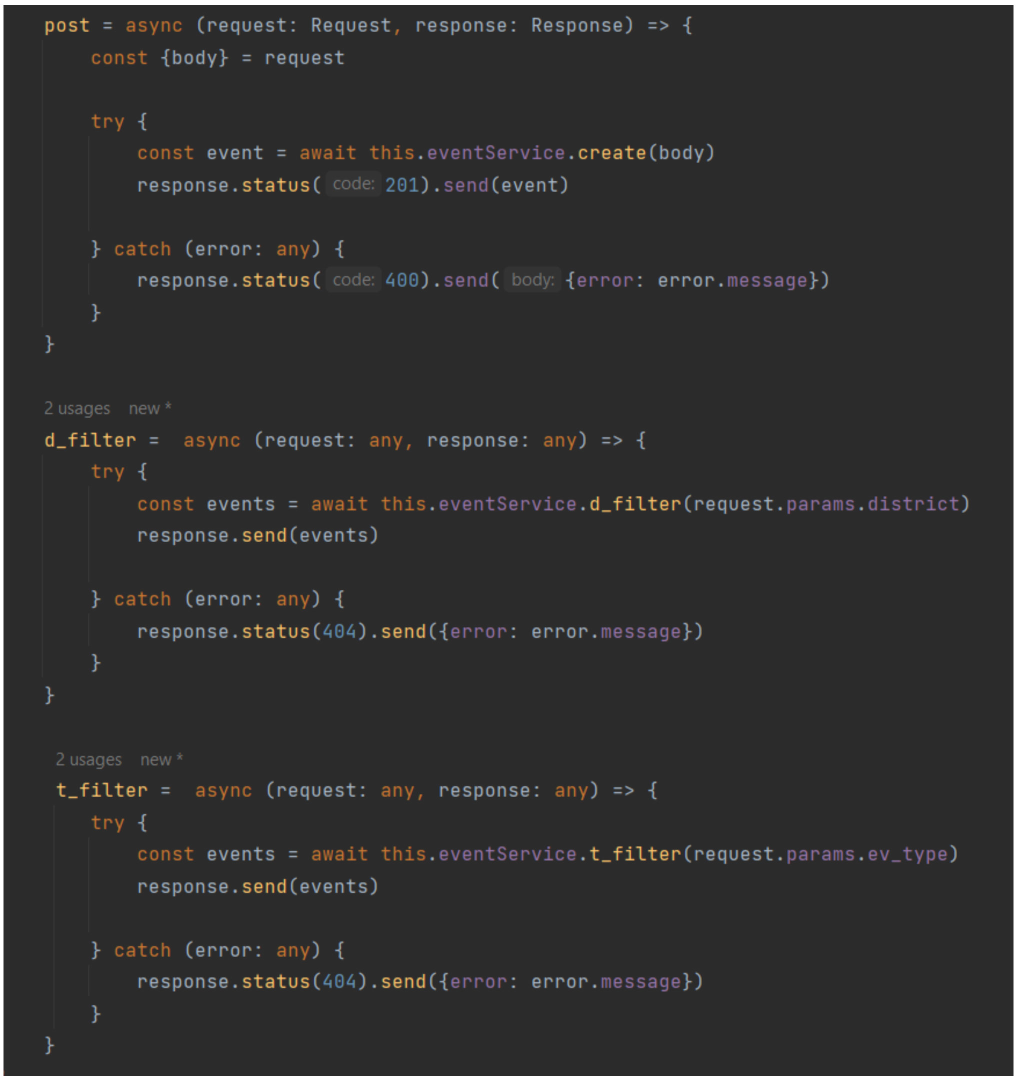


```
    2 usages   new *
    update = async (request: any, response: any) => {
        try {
          const { body } = request;
          const { id } = request.params
          const user = await this.userService.changeUserInfo(Number(id), body);

          response.send(user);
        } catch (error:any) {
          response.status(400).send(error.message);
        }
    }

    3 usages   new *
    delete = async (request: any, response: any) => {
        try {
            await this.userService.deleteUser(request.params.id)
            response.send('User deleted')
        } catch (error: any) {
            response.sendStatus( code: 400)
        }
    }
}
```

Event.ts

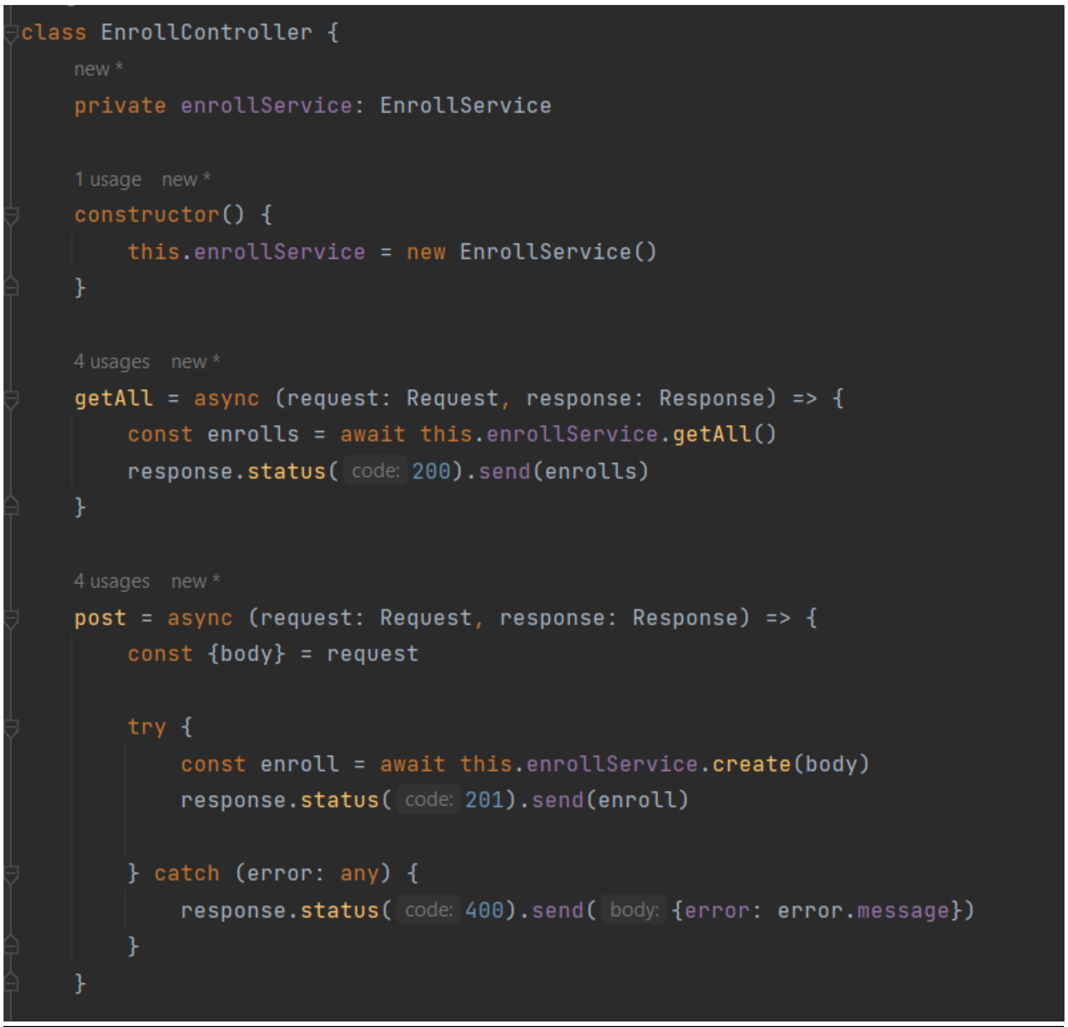


```
class EventController {
    new *
    private eventService: EventService

    1 usage   new *
    constructor() {
        this.eventService = new EventService()
    }

    4 usages   new *
    getAll = async (request: Request, response: Response) => {
        const events = await this.eventService.getAll()
        response.status( code: 200).send(events)
    }

    3 usages   new *
    get = async (request: Request, response: Response) => {
        try {
            const event = await this.eventService.getById(Number(request.params.id))
            response.send(event)

        } catch (error: any) {
            response.status( code: 404).send( body: {error: error.message})
        }
    }
```

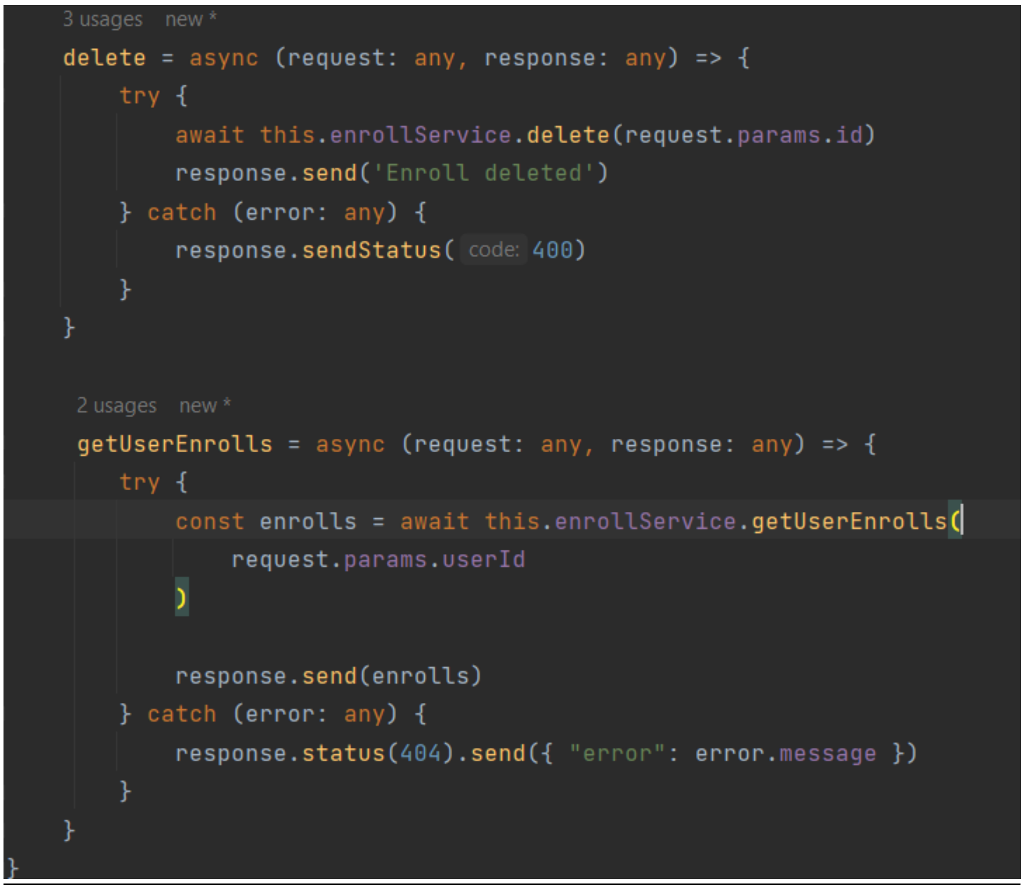


```
post = async (request: Request, response: Response) => {
    const {body} = request

    try {
        const event = await this.eventService.create(body)
        response.status( code: 201).send(event)

    } catch (error: any) {
        response.status( code: 400).send( body: {error: error.message})
    }
}

2 usages   new *
d_filter =  async (request: any, response: any) => {
    try {
        const events = await this.eventService.d_filter(request.params.district)
        response.send(events)

    } catch (error: any) {
        response.status(404).send({error: error.message})
    }
}

 2 usages   new *
 t_filter =  async (request: any, response: any) => {
    try {
        const events = await this.eventService.t_filter(request.params.ev_type)
        response.send(events)

    } catch (error: any) {
        response.status(404).send({error: error.message})
    }
}
```

Enroll.ts



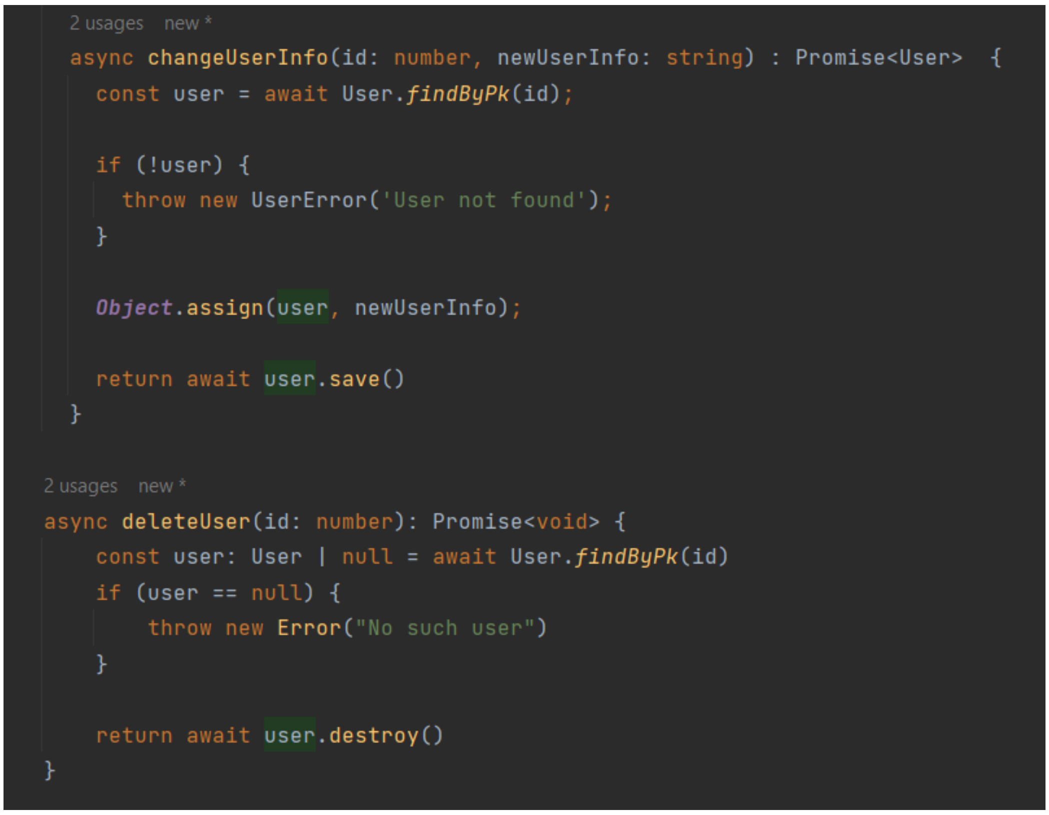
```
class EnrollController {
    new *
    private enrollService: EnrollService

    1 usage   new *
    constructor() {
        this.enrollService = new EnrollService()
    }

    4 usages   new *
    getAll = async (request: Request, response: Response) => {
        const enrolls = await this.enrollService.getAll()
        response.status( code: 200).send(enrolls)
    }

    4 usages   new *
    post = async (request: Request, response: Response) => {
        const {body} = request

        try {
            const enroll = await this.enrollService.create(body)
            response.status( code: 201).send(enroll)

        } catch (error: any) {
            response.status( code: 400).send( body: {error: error.message})
        }
    }
```



```
    3 usages   new *
    delete = async (request: any, response: any) => {
        try {
            await this.enrollService.delete(request.params.id)
            response.send('Enroll deleted')
        } catch (error: any) {
            response.sendStatus( code: 400)
        }
    }

     2 usages   new *
     getUserEnrolls = async (request: any, response: any) => {
        try {
            const enrolls = await this.enrollService.getUserEnrolls(
                request.params.userId
            )

            response.send(enrolls)
        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }
}
```

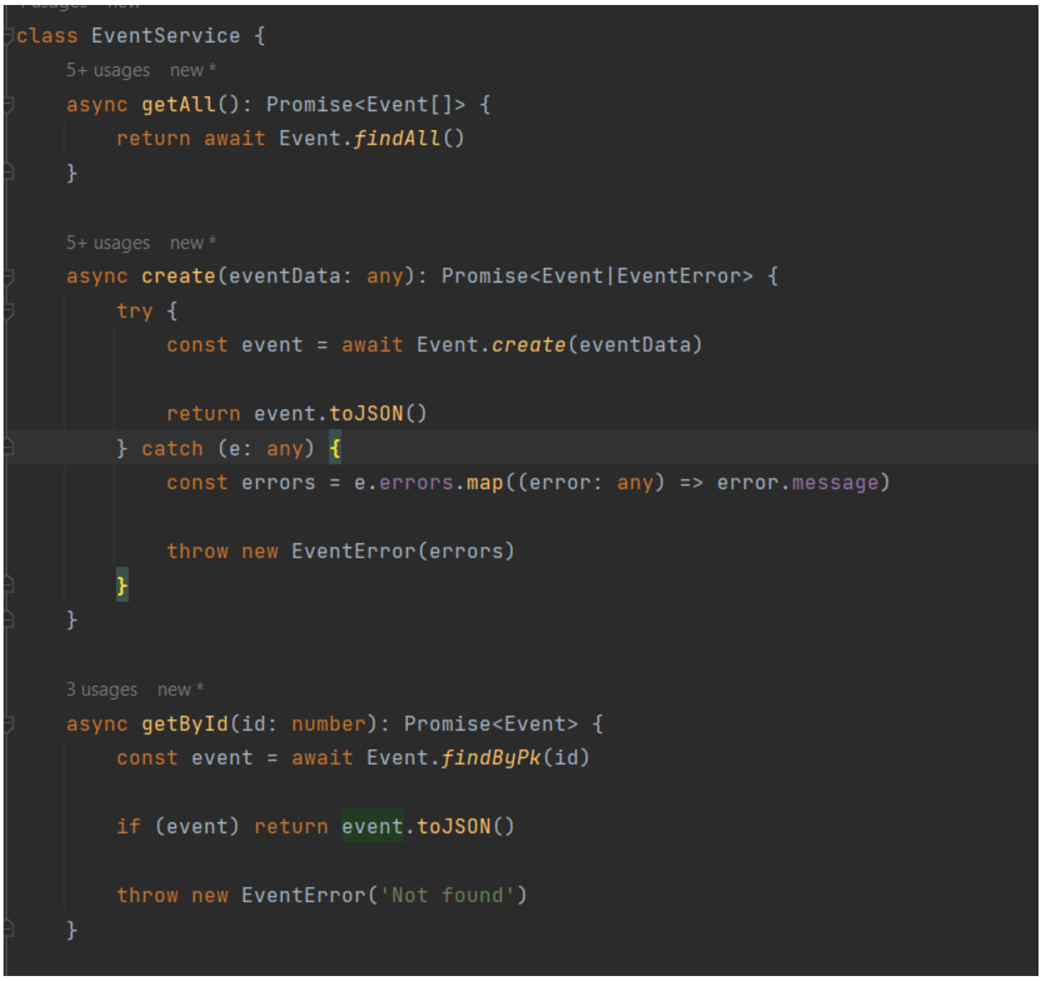
Новые функции в сервисах:

User.ts

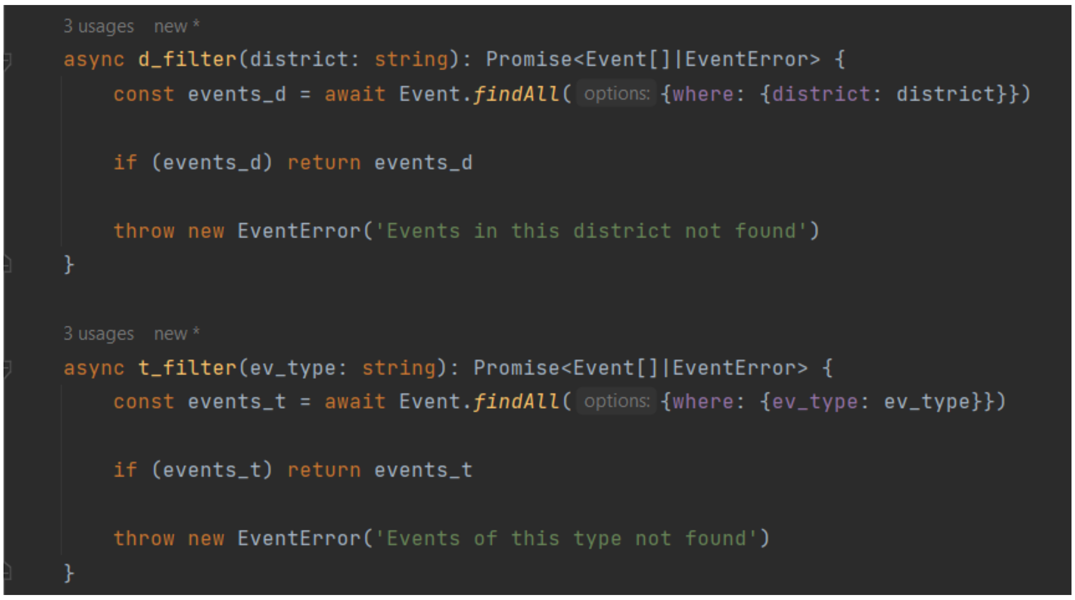


```
  2 usages   new *
  async changeUserInfo(id: number, newUserInfo: string) : Promise<User>  {
    const user = await User.findByPk(id);

    if (!user) {
      throw new UserError('User not found');
    }

    Object.assign(user, newUserInfo);

    return await user.save()
  }

 2 usages   new *
 async deleteUser(id: number): Promise<void> {
     const user: User | null = await User.findByPk(id)
     if (user == null) {
         throw new Error("No such user")
     }

     return await user.destroy()
 }
```

Event.ts

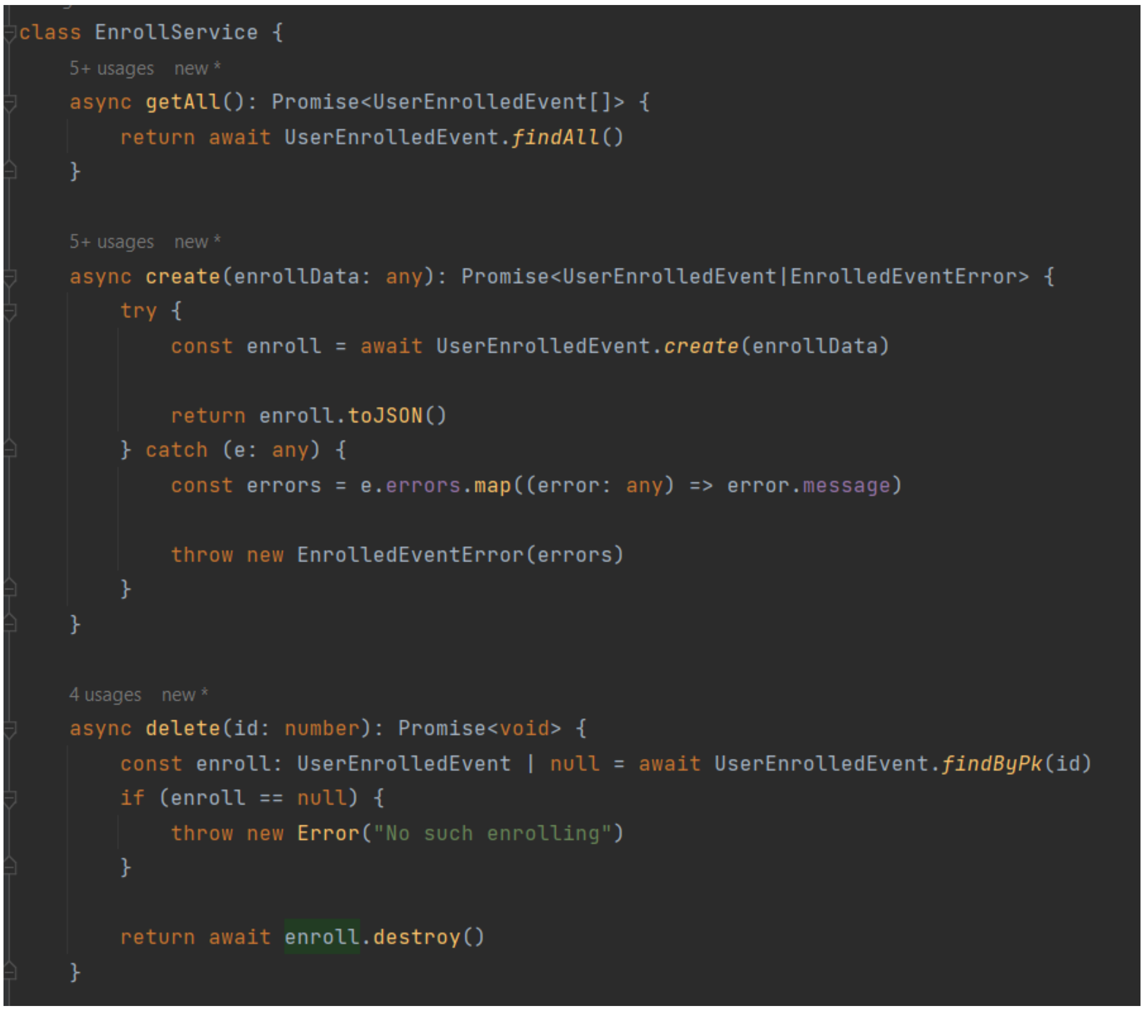


```
class EventService {
    5+ usages   new *
    async getAll(): Promise<Event[]> {
        return await Event.findAll()
    }

    5+ usages   new *
    async create(eventData: any): Promise<Event|EventError> {
        try {
            const event = await Event.create(eventData)

            return event.toJSON()
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)

            throw new EventError(errors)
        }
    }

    3 usages   new *
    async getById(id: number): Promise<Event> {
        const event = await Event.findByPk(id)

        if (event) return event.toJSON()

        throw new EventError('Not found')
    }
```

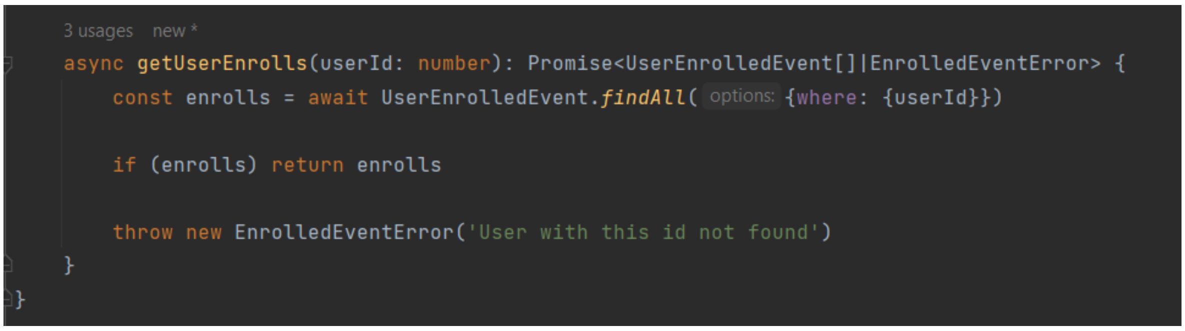


```
3 usages   new *
async d_filter(district: string): Promise<Event[]|EventError> {
    const events_d = await Event.findAll( options: {where: {district: district}})

    if (events_d) return events_d

    throw new EventError('Events in this district not found')
}

3 usages   new *
async t_filter(ev_type: string): Promise<Event[]|EventError> {
    const events_t = await Event.findAll( options: {where: {ev_type: ev_type}})

    if (events_t) return events_t

    throw new EventError('Events of this type not found')
}
```

Enroll.ts



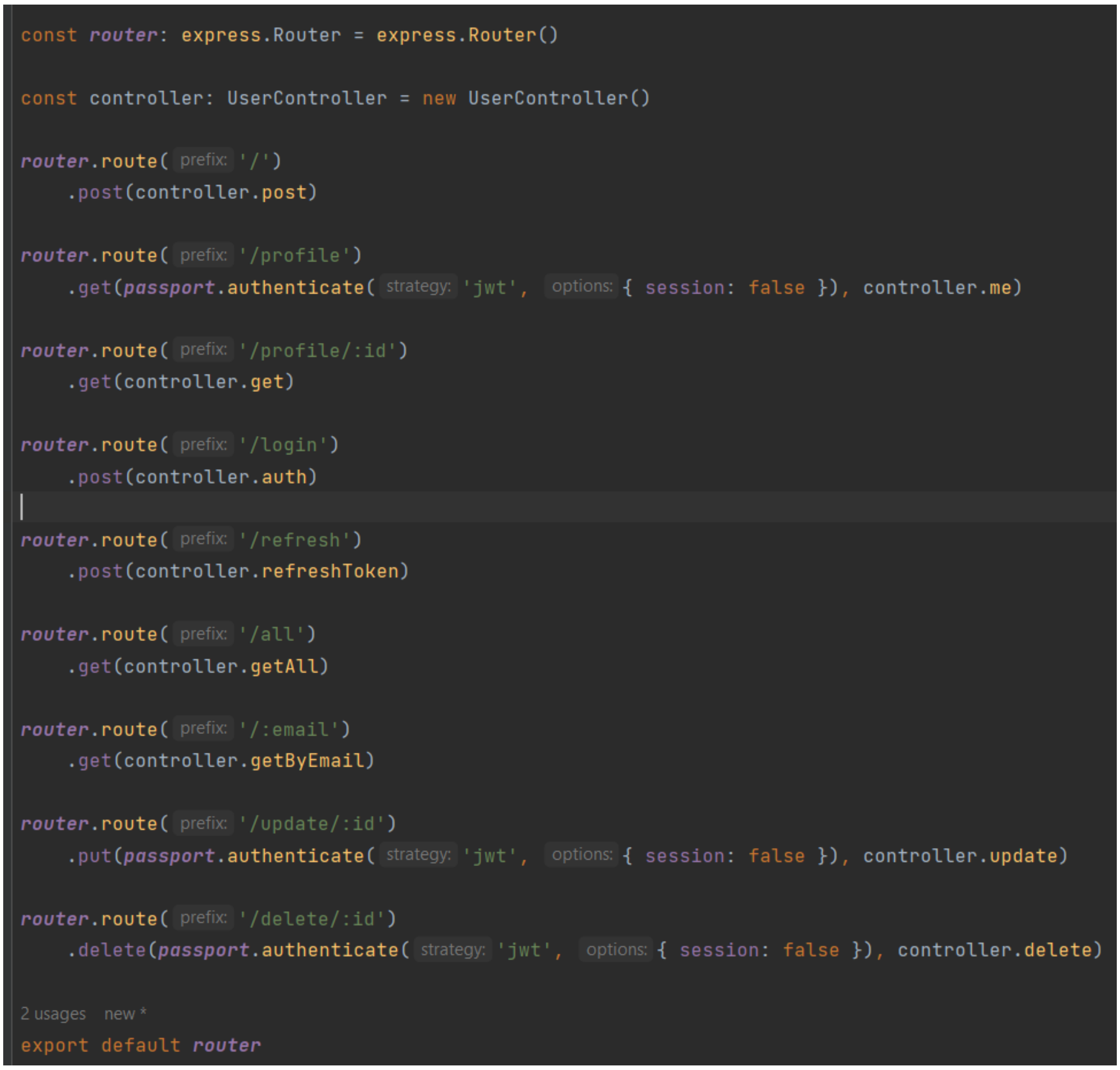
```
class EnrollService {
    5+ usages   new *
    async getAll(): Promise<UserEnrolledEvent[]> {
        return await UserEnrolledEvent.findAll()
    }

    5+ usages   new *
    async create(enrollData: any): Promise<UserEnrolledEvent|EnrolledEventError> {
        try {
            const enroll = await UserEnrolledEvent.create(enrollData)

            return enroll.toJSON()
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)

            throw new EnrolledEventError(errors)
        }
    }

    4 usages   new *
    async delete(id: number): Promise<void> {
        const enroll: UserEnrolledEvent | null = await UserEnrolledEvent.findByPk(id)
        if (enroll == null) {
            throw new Error("No such enrolling")
        }

        return await enroll.destroy()
    }
```



```
    3 usages   new *
    async getUserEnrolls(userId: number): Promise<UserEnrolledEvent[]|EnrolledEventError> {
        const enrolls = await UserEnrolledEvent.findAll( options: {where: {userId}})

        if (enrolls) return enrolls

        throw new EnrolledEventError('User with this id not found')
    }
}
```

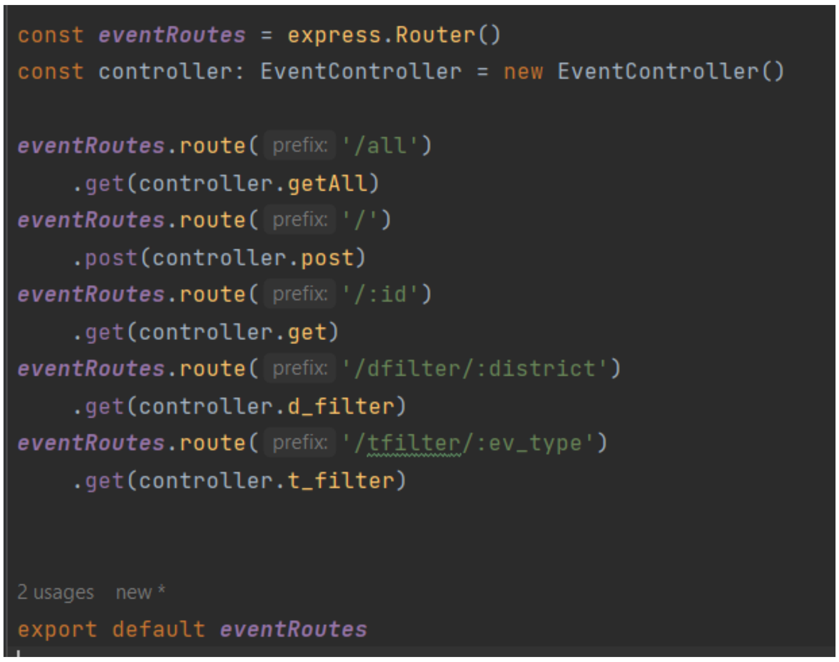
Routes:

User.ts



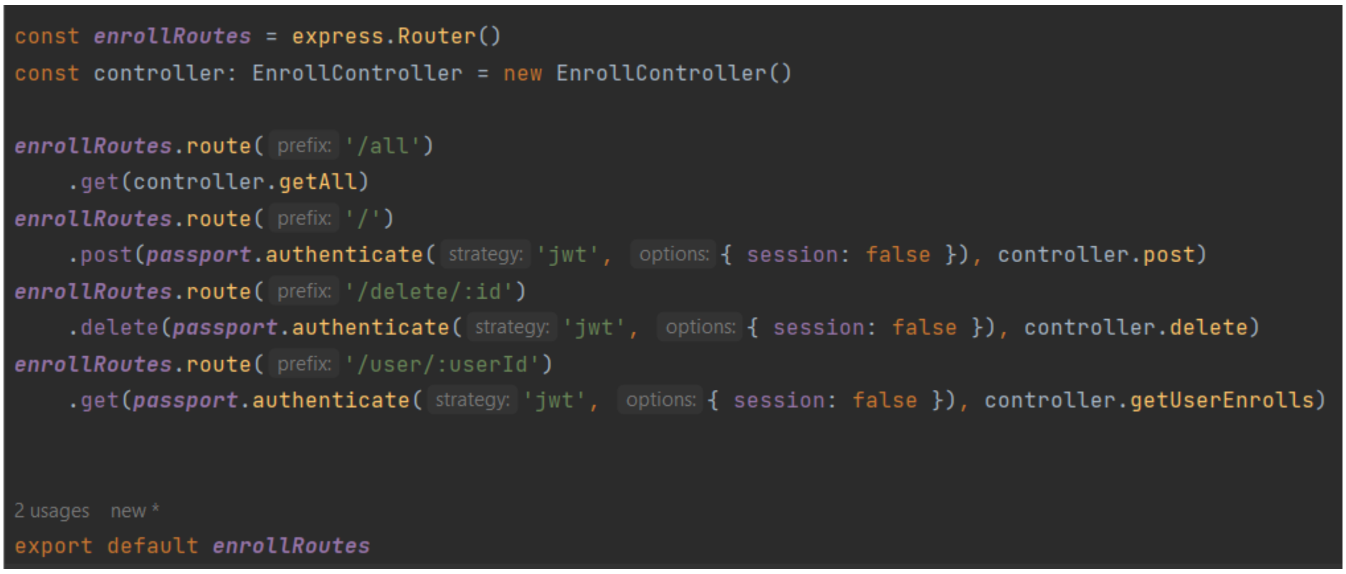
```
const router: express.Router = express.Router()

const controller: UserController = new UserController()

router.route( prefix: '/')
    .post(controller.post)

router.route( prefix: '/profile')
    .get(passport.authenticate( strategy: 'jwt',  options: { session: false }), controller.me)

router.route( prefix: '/profile/:id')
    .get(controller.get)

router.route( prefix: '/login')
    .post(controller.auth)

router.route( prefix: '/refresh')
    .post(controller.refreshToken)

router.route( prefix: '/all')
    .get(controller.getAll)

router.route( prefix: '/:email')
    .get(controller.getByEmail)

router.route( prefix: '/update/:id')
    .put(passport.authenticate( strategy: 'jwt',  options: { session: false }), controller.update)

router.route( prefix: '/delete/:id')
    .delete(passport.authenticate( strategy: 'jwt',  options: { session: false }), controller.delete)

2 usages   new *
export default router
```

Event.ts



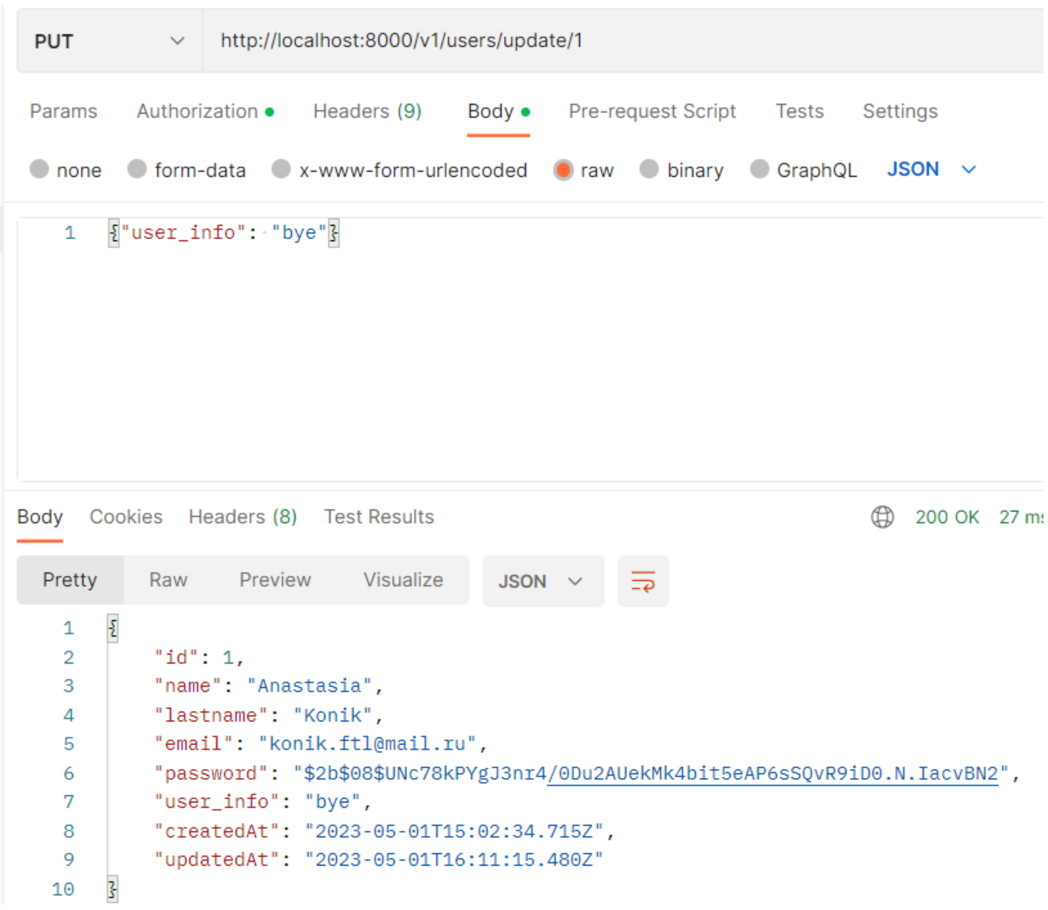
```
const eventRoutes = express.Router()
const controller: EventController = new EventController()

eventRoutes.route( prefix: '/all')
    .get(controller.getAll)
eventRoutes.route( prefix: '/')
    .post(controller.post)
eventRoutes.route( prefix: '/:id')
    .get(controller.get)
eventRoutes.route( prefix: '/dfilter/:district')
    .get(controller.d_filter)
eventRoutes.route( prefix: '/tfilter/:ev_type')
    .get(controller.t_filter)


2 usages   new *
export default eventRoutes
```

Enroll.ts



```
const enrollRoutes = express.Router()
const controller: EnrollController = new EnrollController()

enrollRoutes.route( prefix: '/all')
    .get(controller.getAll)
enrollRoutes.route( prefix: '/')
    .post(passport.authenticate( strategy: 'jwt', options: { session: false }), controller.post)
enrollRoutes.route( prefix: '/delete/:id')
    .delete(passport.authenticate( strategy: 'jwt', options: { session: false }), controller.delete)
enrollRoutes.route( prefix: '/user/:userId')
    .get(passport.authenticate( strategy: 'jwt', options: { session: false }), controller.getUserEnrolls)


2 usages   new *
export default enrollRoutes
```

Обновление информации о пользователе:



PUT http://localhost:8000/v1/users/update/1

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON

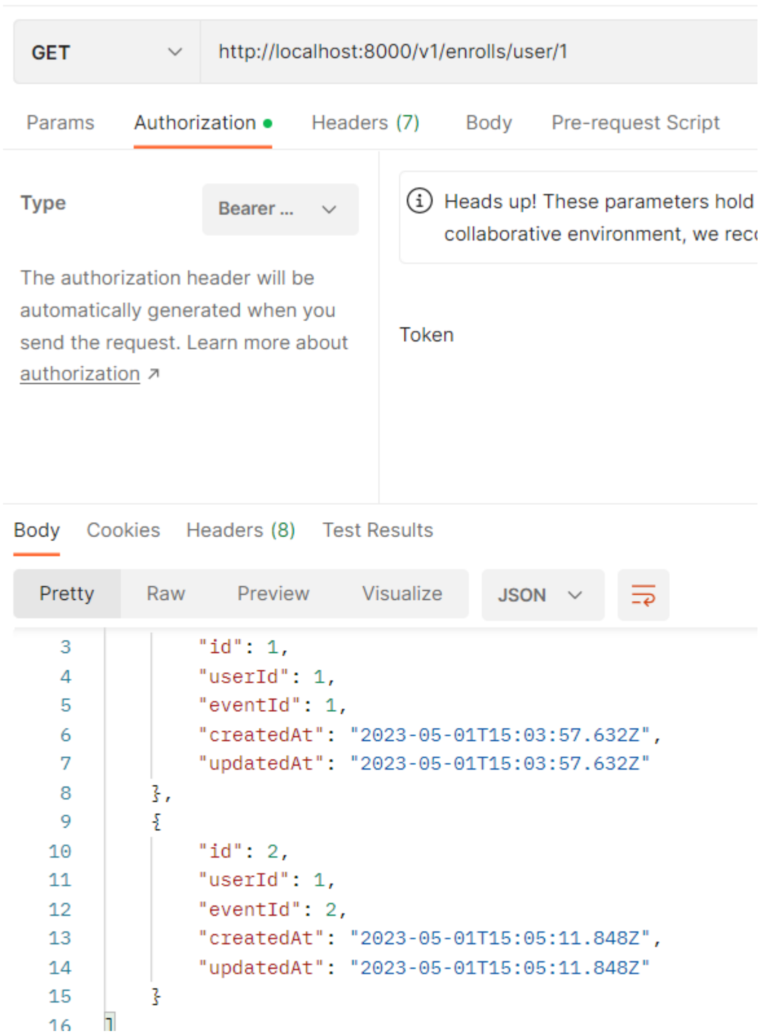
```
{"user_info": "bye"}
```

Body | Cookies | Headers (8) | Test Results   200 OK   27 m

Pretty | Raw | Preview | Visualize | JSON

```
{
    "id": 1,
    "name": "Anastasia",
    "lastname": "Konik",
    "email": "konik.ftl@mail.ru",
    "password": "$2b$08$UNc78kPYgJ3nr4/0Du2AUekMk4bit5eAP6sSQvR9iD0.N.IacvBN2",
    "user_info": "bye",
    "createdAt": "2023-05-01T15:02:34.715Z",
    "updatedAt": "2023-05-01T16:11:15.480Z"
}
```

Мероприятия пользователя:



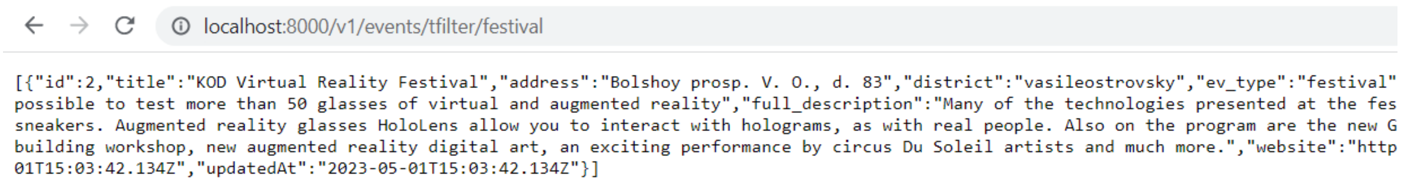
http://localhost:8000/v1/enrolls/user/1

Heads up! These parameters hold
collaborative environment, we rec

The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

```
            "id": 1,
            "userId": 1,
            "eventId": 1,
            "createdAt": "2023-05-01T15:03:57.632Z",
            "updatedAt": "2023-05-01T15:03:57.632Z"
        },
        {
            "id": 2,
            "userId": 1,
            "eventId": 2,
            "createdAt": "2023-05-01T15:05:11.848Z",
            "updatedAt": "2023-05-01T15:05:11.848Z"
        }
    ]
```

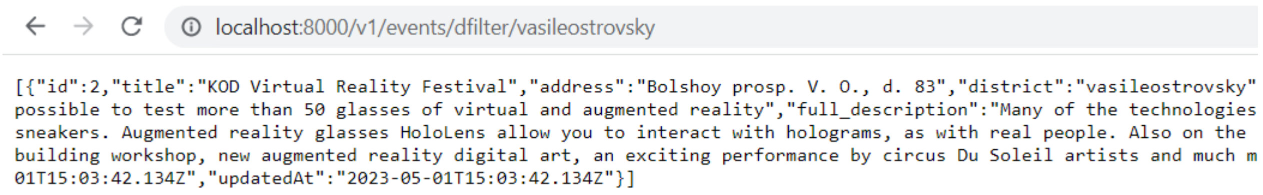
Фильтрация по типу мероприятия:



localhost:8000/v1/events/tfilter/festival

```
[{"id":2,"title":"KOD Virtual Reality Festival","address":"Bolshoy prosp. V. O., d. 83","district":"vasileostrovsky","ev_type":"festival"
possible to test more than 50 glasses of virtual and augmented reality","full_description":"Many of the technologies presented at the fes
sneakers. Augmented reality glasses HoloLens allow you to interact with holograms, as with real people. Also on the program are the new G
building workshop, new augmented reality digital art, an exciting performance by circus Du Soleil artists and much more.","website":"http
01T15:03:42.134Z","updatedAt":"2023-05-01T15:03:42.134Z"}]
```

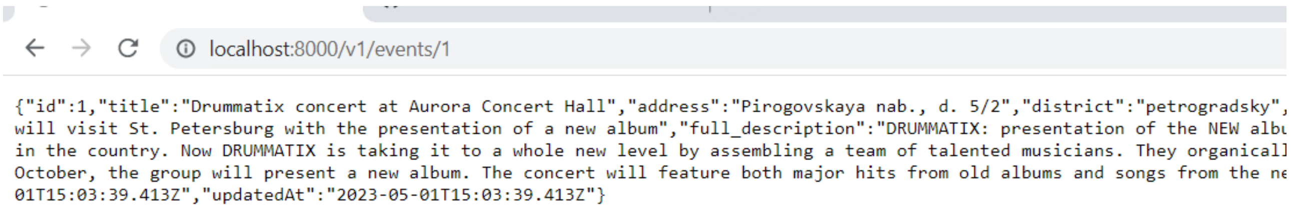
Фильтрация по району:



localhost:8000/v1/events/dfilter/vasileostrovsky

```
[{"id":2,"title":"KOD Virtual Reality Festival","address":"Bolshoy prosp. V. O., d. 83","district":"vasileostrovsky"
possible to test more than 50 glasses of virtual and augmented reality","full_description":"Many of the technologies
sneakers. Augmented reality glasses HoloLens allow you to interact with holograms, as with real people. Also on the
building workshop, new augmented reality digital art, an exciting performance by circus Du Soleil artists and much m
01T15:03:42.134Z","updatedAt":"2023-05-01T15:03:42.134Z"}]
```

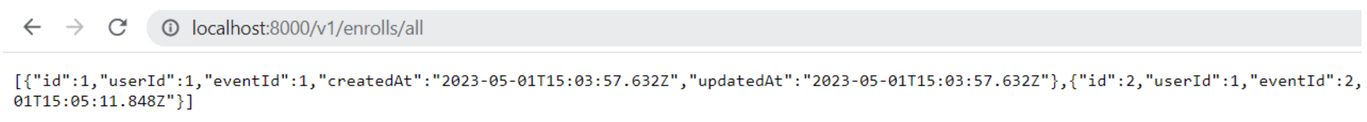
Получение мероприятия:



localhost:8000/v1/events/1

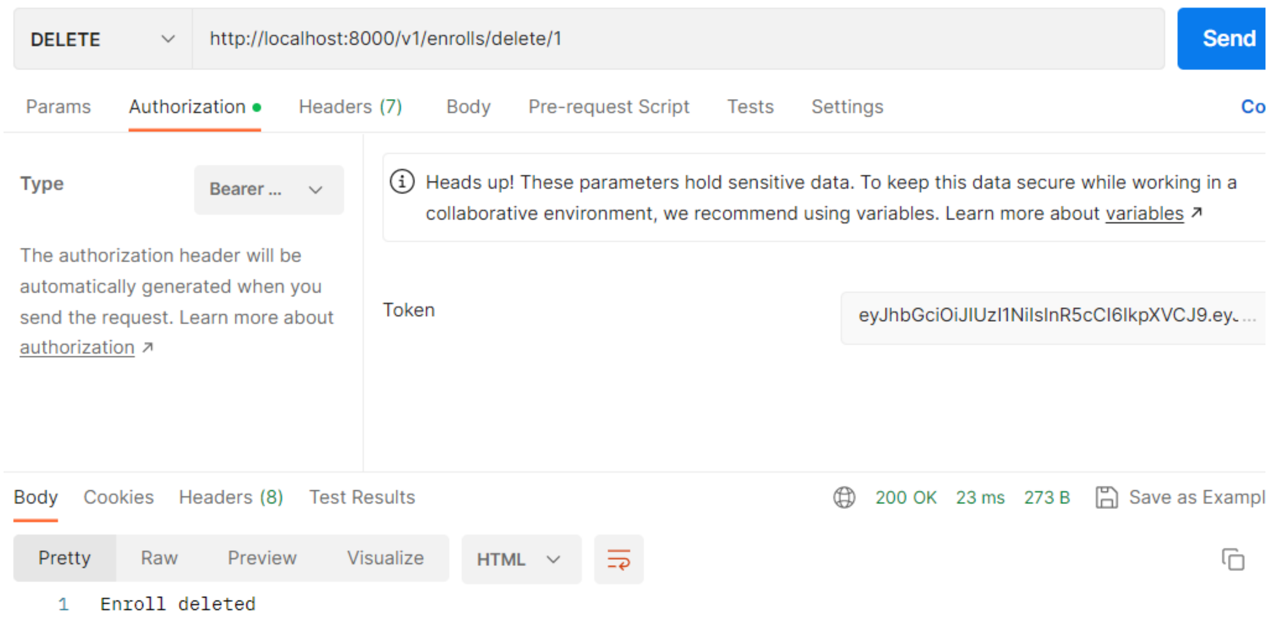
```
{"id":1,"title":"Drummatix concert at Aurora Concert Hall","address":"Pirogovskaya nab., d. 5/2","district":"petrogradsky",
will visit St. Petersburg with the presentation of a new album","full_description":"DRUMMATIX: presentation of the NEW albu
in the country. Now DRUMMATIX is taking it to a whole new level by assembling a team of talented musicians. They organicall
October, the group will present a new album. The concert will feature both major hits from old albums and songs from the ne
01T15:03:39.413Z","updatedAt":"2023-05-01T15:03:39.413Z"}
```

Все записи:


localhost:8000/v1/enrolls/all

```
[{"id":1,"userId":1,"eventId":1,"createdAt":"2023-05-01T15:03:57.632Z","updatedAt":"2023-05-01T15:03:57.632Z"},{"id":2,"userId":1,"eventId":2,
01T15:05:11.848Z"}]
```

Удаление записи пользователя:


DELETE http://localhost:8000/v1/enrolls/delete/1

Type: Bearer ...

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables ↗

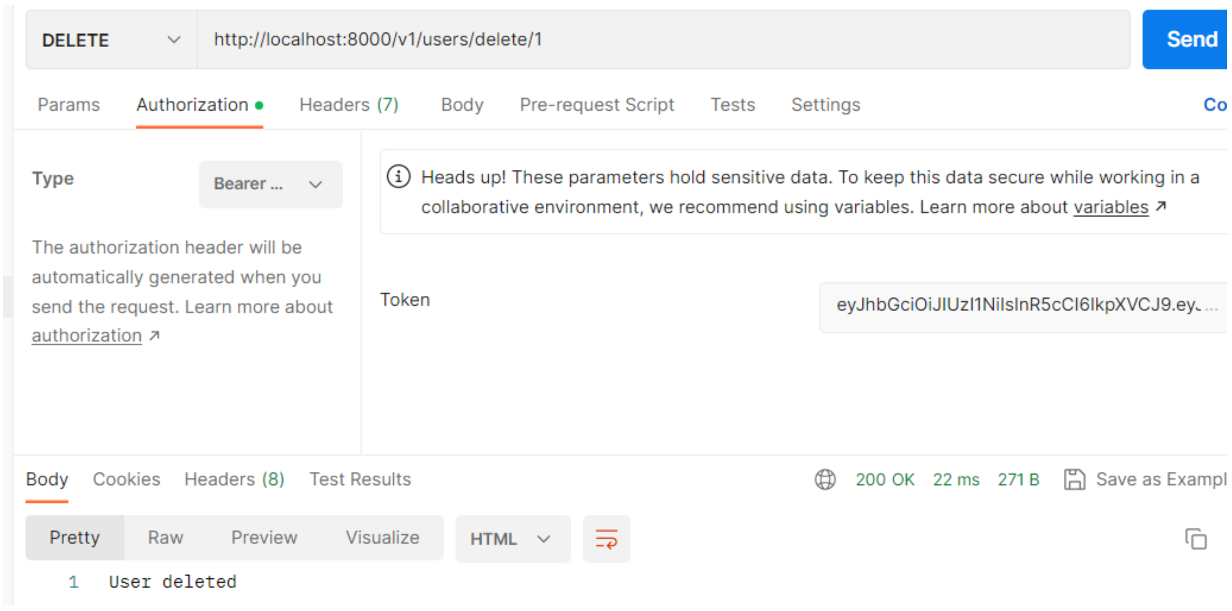
The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey...

200 OK 23 ms 273 B

```
Enroll deleted
```

Удаление пользователя:


DELETE http://localhost:8000/v1/users/delete/1

Type: Bearer ...

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables ↗

The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey...

200 OK 22 ms 271 B

```
User deleted
```

Вывод

В ходе лабораторной работы был реализован RESTful API средствами express + typescript (используя ранее написанный boilerplate) для поиска мероприятий в Санкт-Петербурге.