САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа № 2

“Тестирование, разработка и документирование RESTful API”

Выполнил:
Ле Хоанг Чыонг
Чан Дык Минь

Группа:
K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

Нужно реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate).

Вариант №3: Сервис-помощник в планировании путешествий.

Требуемый функционал:

- регистрация, авторизация,
- создание профиля,
- создание поездки с заполнением информации о ней,
- поиск по существующим предложениям для поездок,
- предложения для новой созданной поездки на основе интересов пользователя.

Ход работы

1. Проект будет иметь следующую структуру:

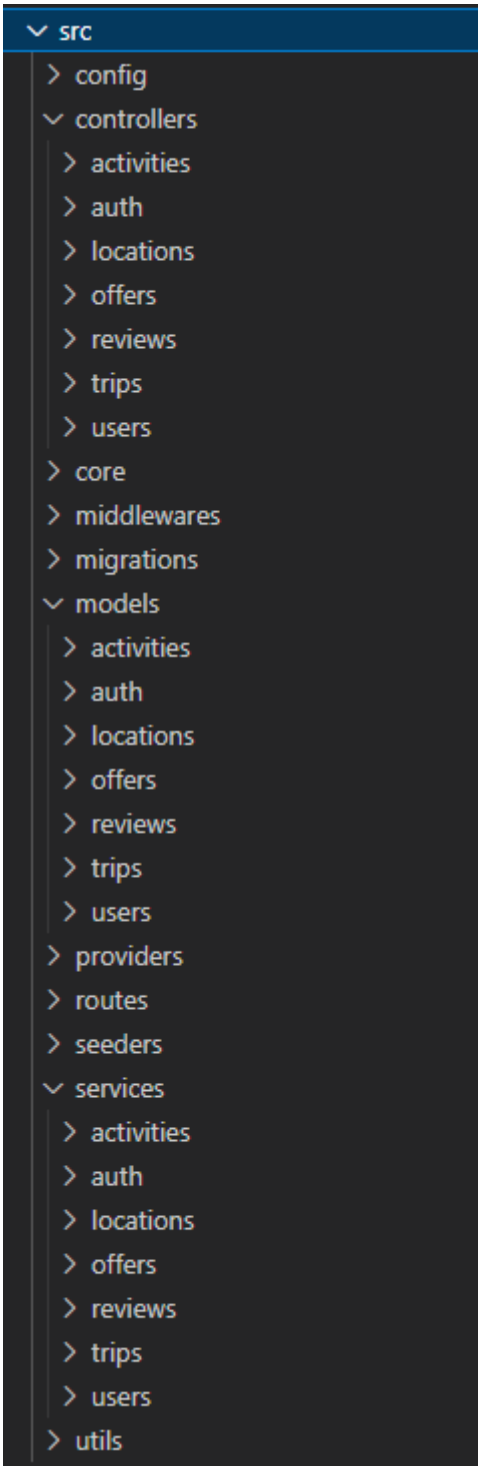


Рисунок 1 — Файловая структура проекта

2. Вход и регистрация были реализованы еще в первой лабораторной работе.

3. Создание профиля.

Часть кода из файла src>models>users>user.ts:

```
@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)
        }
    }

    @BelongsToMany(() => Activity, () => UserActivity)
    activities: Activity[];

    @BelongsToMany(() => Location, () => Review)
    locations: Location[];
}

export default User
```

Рисунок 2 — Модель User

Часть кода из файла src>controllers>auth>auth.ts:

```
register = async (request: any, response: any) => {
    const { body } = request
    try {
        const data: any = await this.authService.register(body)
        response.status(201).send(data)
    } catch (error: any) {
        response.status(400).send(getErrorMessage(error))
    }
}
```

Рисунок 3 — Auth Controller

```
createUserActivity = async (request: any, response: any) => {
    try {
        const { body } = request;
        const userId = request.user.id;
        const data: any = await this.userActivityService.create({ ...body, user_id: userId })
        response.status(201).send(data)
    } catch (error: any) {
        response.status(404).send(getErrorMessage(error))
    }
}
```

Рисунок 4 — UserActivity Controller

4. Создание поездки с заполнением информации о ней

Часть кода из файла src>models>trips>trip.ts:

```ts
@Table
export class Trip extends Model<TripAttributes, TripCreationAttributes> {
  @PrimaryKey
  @Column({
    type: DataType.UUID,
    defaultValue: DataType.UUIDV4,
  })
  id: string;

  @Column
  name: string;

  @Column
  description: string;

  @Column({ type: DataType.DATE })
  dateStart: Date;

  @Column({ type: DataType.DATE })
  dateEnd: Date;

  @Column
  budget: number;

  @Column({
    type: DataType.ENUM(...Object.values(TripType)),
  })
  type: TripType;

  @ForeignKey(() => User)
  @Column({
    type: DataType.UUID,
  })
  userId: string;

  @BelongsTo(() => User)
  user: User;

  @BelongsToMany(() => Location, () => TripLocation)
    locations: Location[];
}
```

Рисунок 5 — Модель Trip

Часть кода из файла src>controllers>trips>trip.ts:

```
create = async (request: any, response: any) => {
    try {
        const { body } = request;
        const userId = request.user.id;
        const trip: any = await this.tripService.create({ ...body, userId: userId });
        response.status(201).send(trip)
    } catch (error: any) {
        response.status(404).send(getErrorMessage(error))
    }
}
```

Рисунок 6 — Controller Trip

Часть кода из файла src>services>trips>trip.ts:

```
async create(tripData: any): Promise<Trip> {
    try {
        const trip = await Trip.create(tripData);
        return trip;
    } catch (error) {
        throw error;
    }
}
```

Рисунок 7 — Trip service

5. Поиск по существующим предложениям для поездок

Каждому пользователю при регистрации необходимо будет сохранить свои любимые занятия, затем на основе этих интересов (например: поход на пляж, восхождение в горы, еды и т. д.) программа найдет место, где есть эти занятия, оттуда вы найдете из предложений этого места

Часть кода из файла src>services>offers>offers.ts:

```
async getOfferForUser(user_id: string, type?: string, price?: number): Promise<Offer[]> {
    try {
        const activities = await UserActivity.findAll({ where: { user_id: user_id } });

        if (!activities || activities.length === 0) {
            return [];
        }
        const activityIds = activities.map(activity => activity.activity_id);

        const locationSet: Set<string> = new Set();

        const locationPromises = activityIds.map(async activityId => {
            const locationsWithActivity = await LocationActivity.findAll({ where: { activity_id: activityId } });
            locationsWithActivity.forEach(location => {
                locationSet.add(location.location_id);
            });
        });
        await Promise.all(locationPromises);

        const locationIds: string[] = [...locationSet];

        const offerPromises: Promise<Offer[]>[] = [];

        locationIds.forEach(locationId => {
            let whereCondition: any = { location_id: locationId };
            if (type) {
                whereCondition.type = type;
            }

            if (price) {
                whereCondition.price = { [Op.lte]: price };
            }

            const offerPromise = Offer.findAll({ where: whereCondition });
            offerPromises.push(offerPromise);
        });

        const allOffers = await Promise.all(offerPromises);

        const flattenedOffers = allOffers.flat();
        return flattenedOffers;
    } catch (error) {
        throw error;
    }
}
```

Рисунок 8 — Offer service

Часть кода из файла src>controllers>offers>offers.ts:

```
getOfferForUser = async (request: any, response: any) => {
    try {
        const user_id = request.user.id;
        const { type, max_price } = request.body;

        const offers: any = await this.offerService.getOfferForUser(user_id, type, max_price);

        response.status(200).send(offers);
    } catch (error: any) {
        response.status(404).send(getErrorMessage(error));
    }
}
```

Рисунок 8 — Offer controller

6. Предложения для новой созданной поездки на основе интересов пользователя.

Так же, как и при поиске offer для пользователей, программа на основе любимых занятий пользователя найдет любимые места, а затем вернет данные пользователю.

Часть кода из файла src>controllers>users>user.ts:

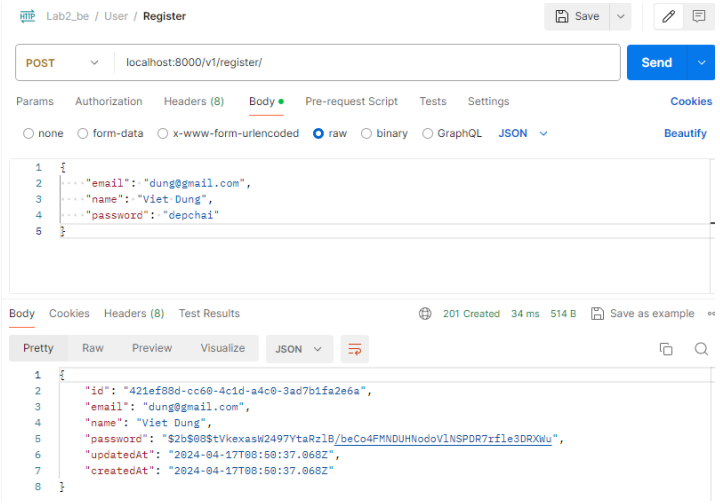
```
async getLocationForUser(user_id: string, rating?: number): Promise<Location[]> {
    try {
        const activities = await UserActivity.findAll({ where: { user_id: user_id } });

        if (!activities || activities.length === 0) {
            return [];
        }

        const activityIds = activities.map(activity => activity.activity_id);

        const locationSet: Set<string> = new Set();

        const locationPromises = activityIds.map(async activityId => {
            const locationsWithActivity = await LocationActivity.findAll({ where: { activity_id: activityId } });
            locationsWithActivity.forEach(location => {
                locationSet.add(location.location_id);
            });
        });

        await Promise.all(locationPromises);

        const locationIds: string[] = [...locationSet];
        let locations: Location[];

        if (rating) {
            locations = await Location.findAll({
                where: {
                    id: locationIds,
                    rating: { [Op.gte]: rating }
                },
                order: [['rating', 'DESC']],
            });
        } else {
            locations = await Location.findAll({
                where: { id: locationIds },
                order: [['rating', 'DESC']],
            });
        }

        return locations;
    } catch (error) {
        throw error;
    }
}
```

Часть кода из файла src>services>users>user.ts:

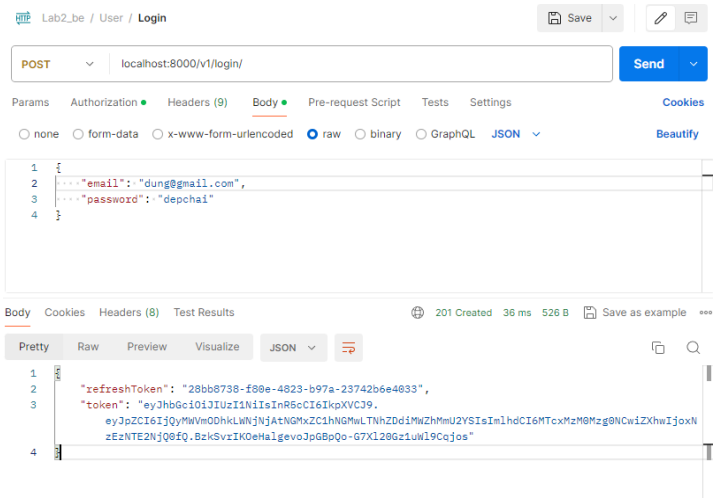
```
getLocationForUser = async (request: any, response: any) => {
    try {
        const user_id = request.user.id;
        const { min_rating } = request.body;
        const locations: Location[] = await this.userService.getLocationForUser(user_id, min_rating);
        response.status(201).send(locations)
    } catch (error: any) {
        response.status(404).send(getErrorMessage(error))
    }
}
```

Postman

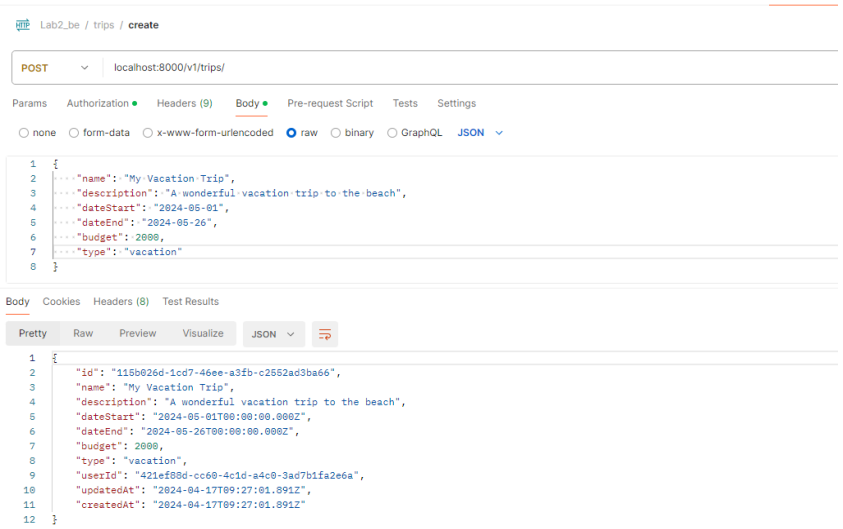
Зарегистрировать нового пользователя



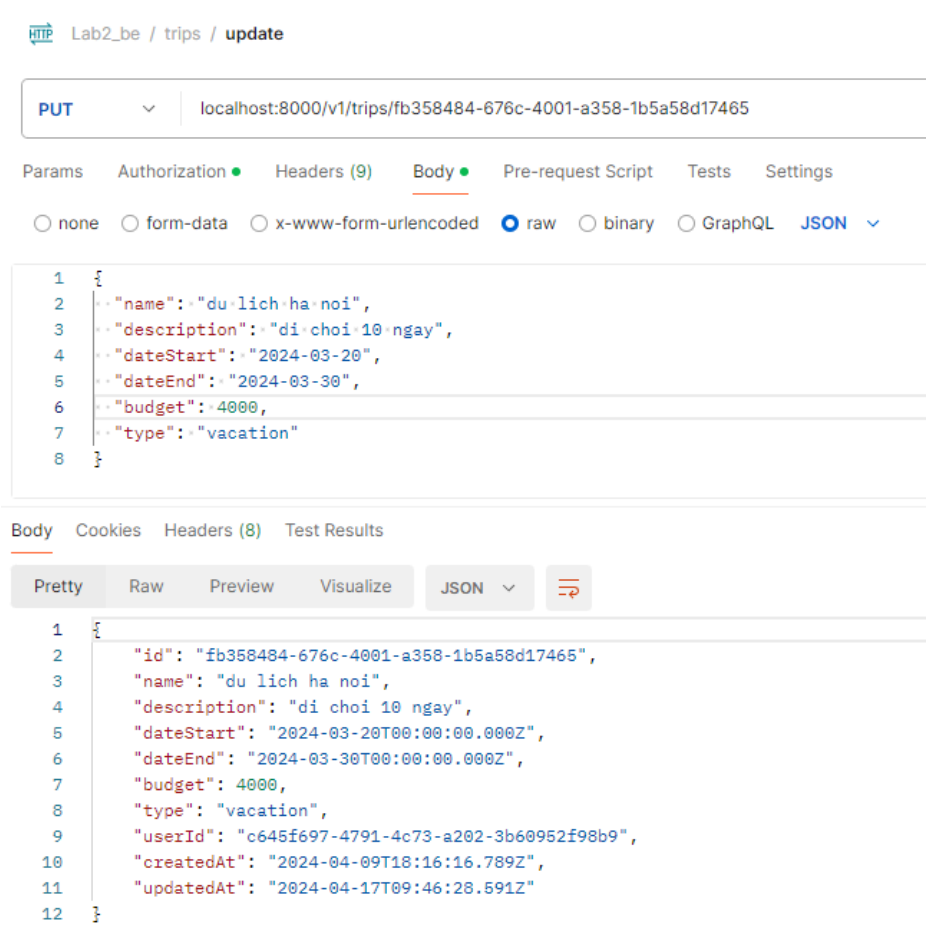
Авторизоваться



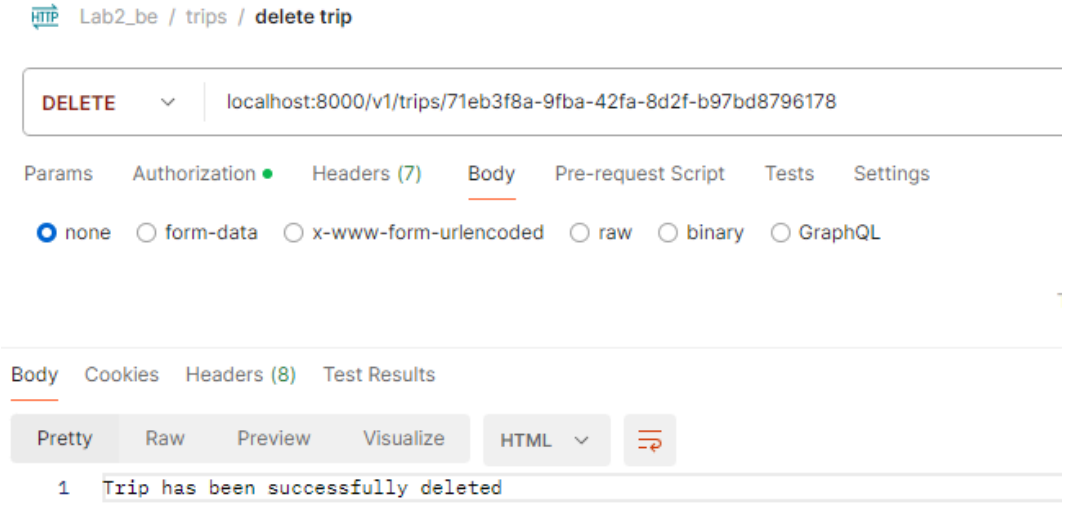
Создать новую поездку



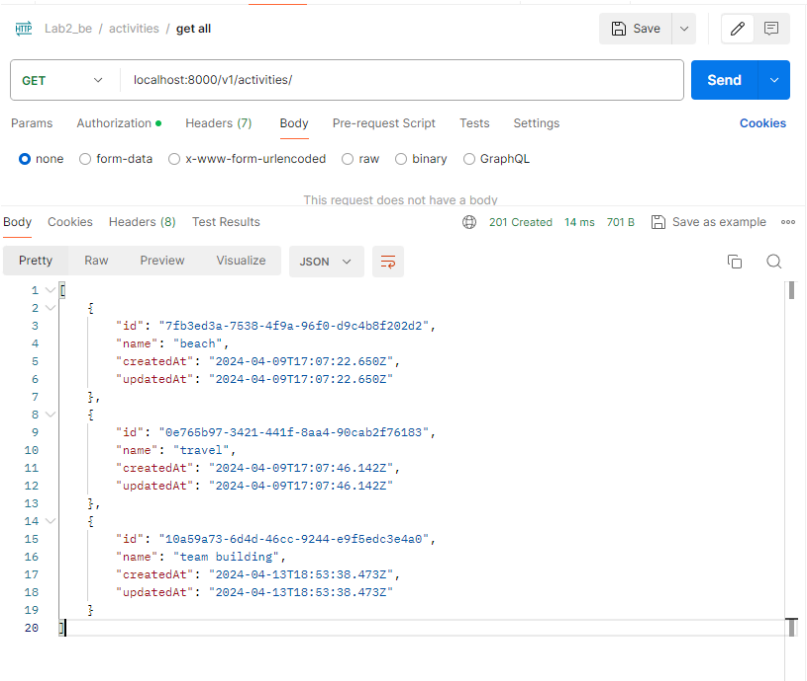
Обновить поездку



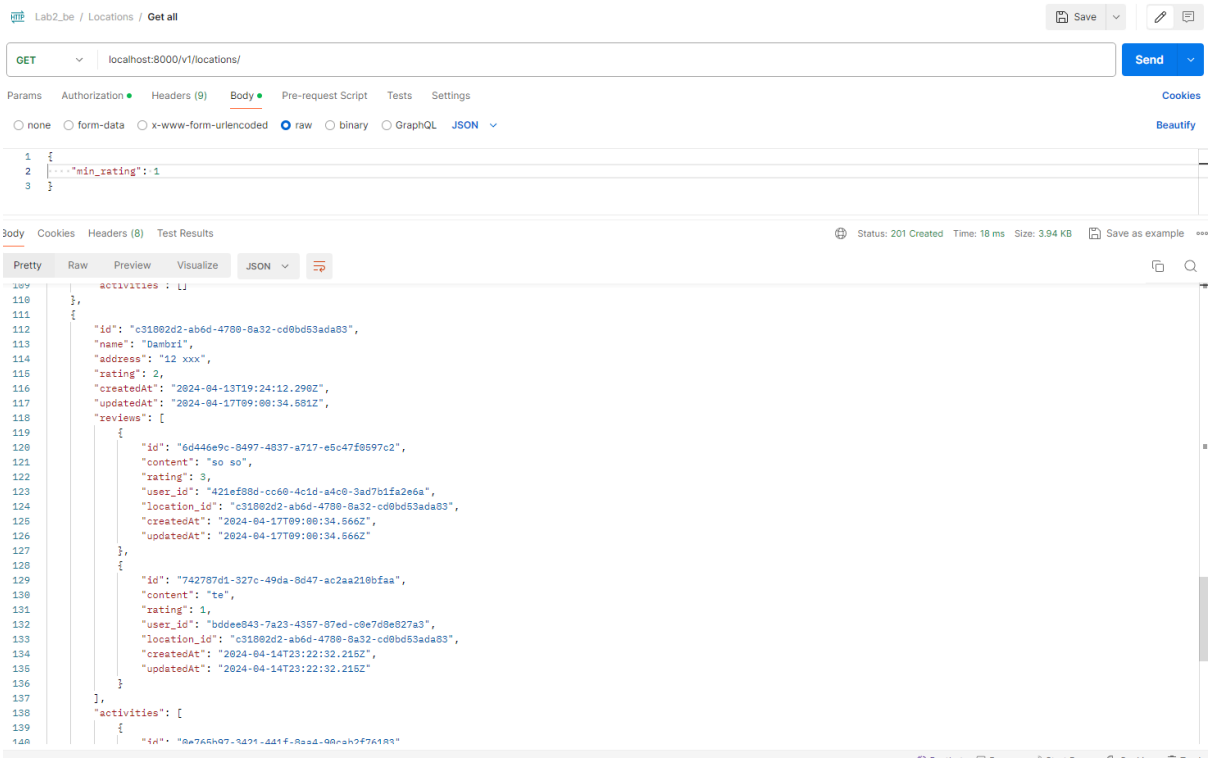
Удалить поездку



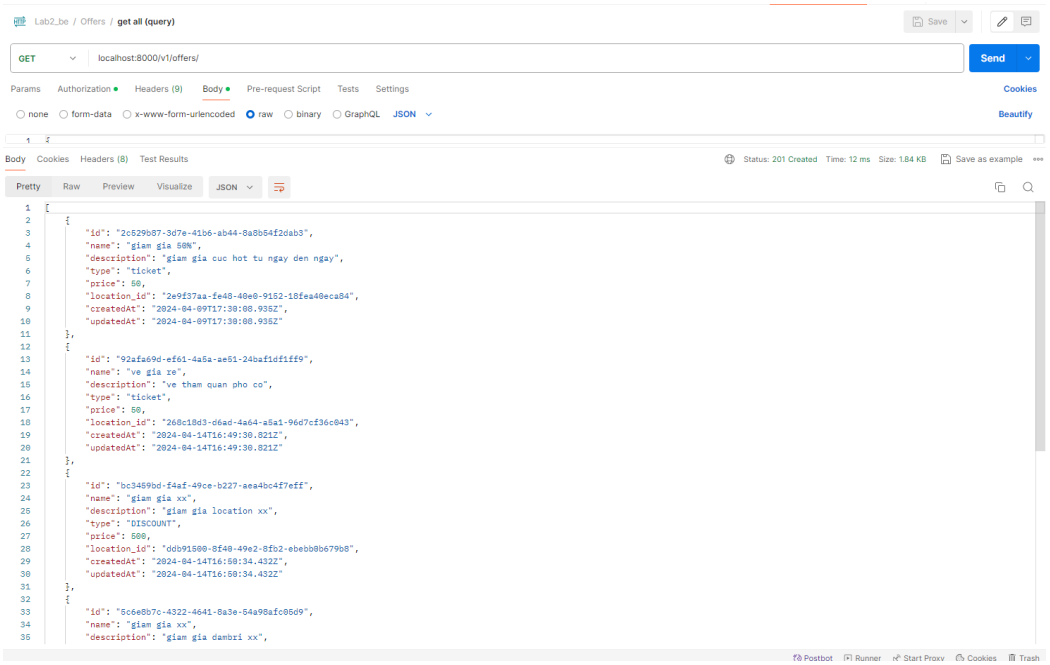
Получить activities (варианты по интересам пользователя)



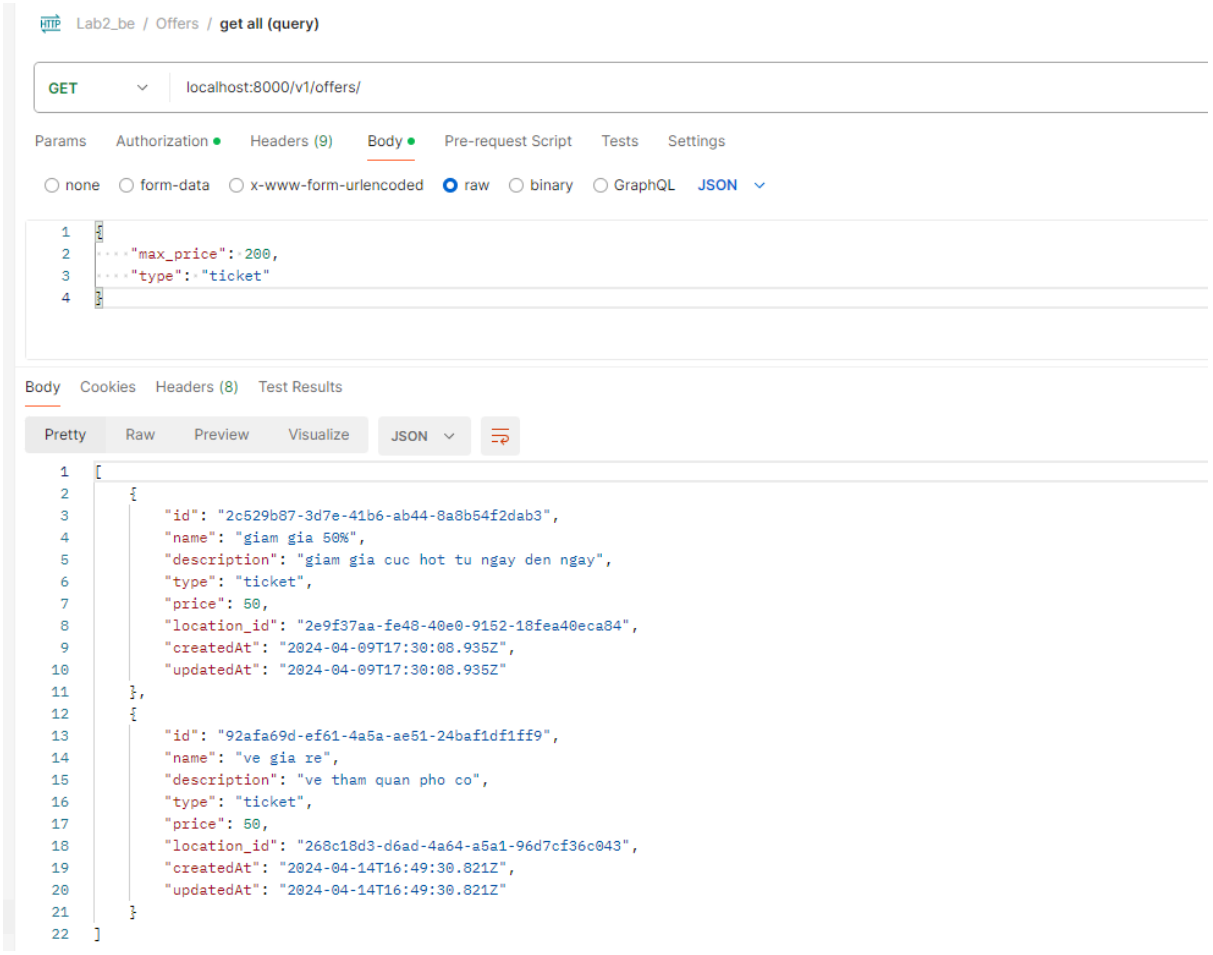
Получить locations, locations ответа будет вложено в reviews и activities.



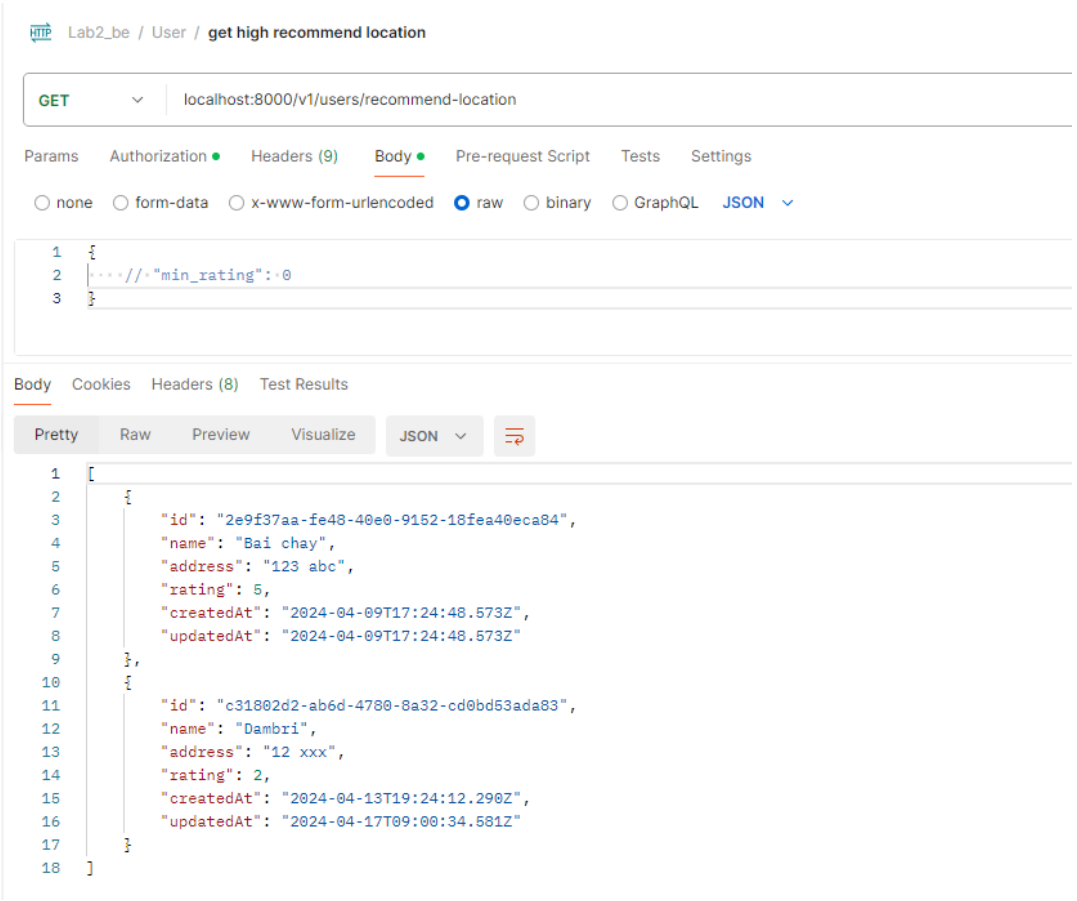
Получить все предложения



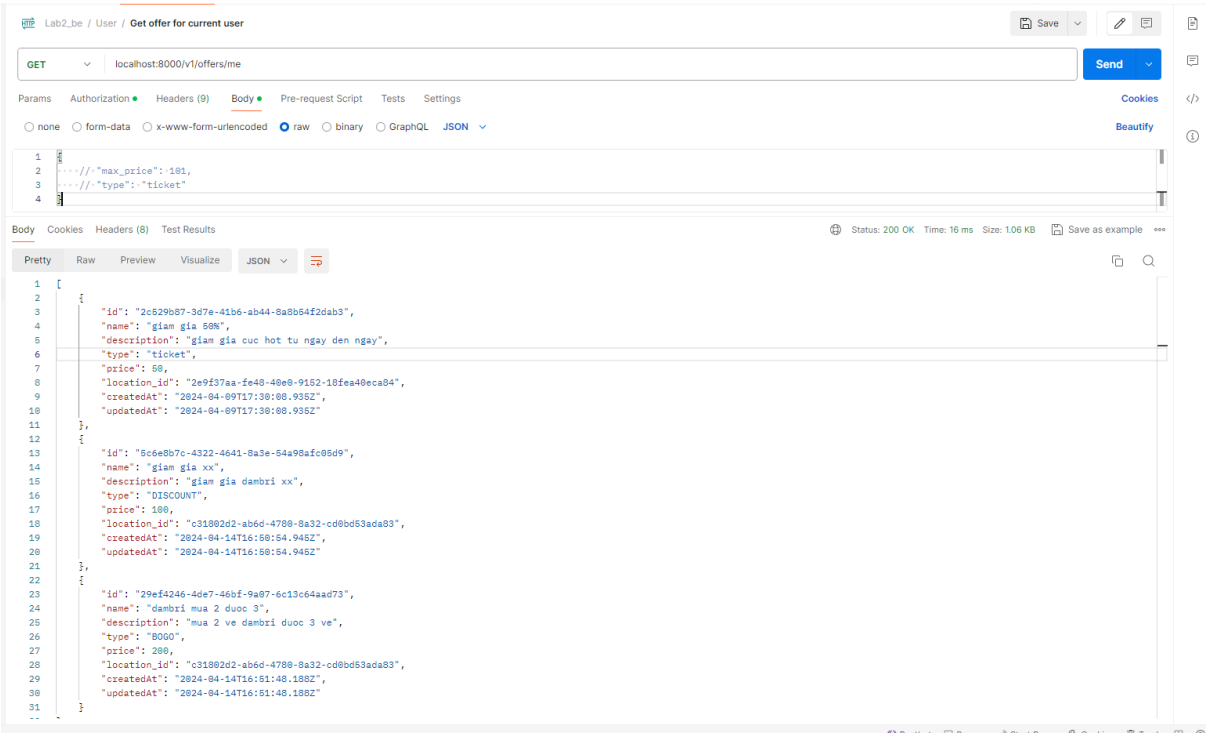
Получить предложение по запросам (цена ниже 200, тип «билет»)



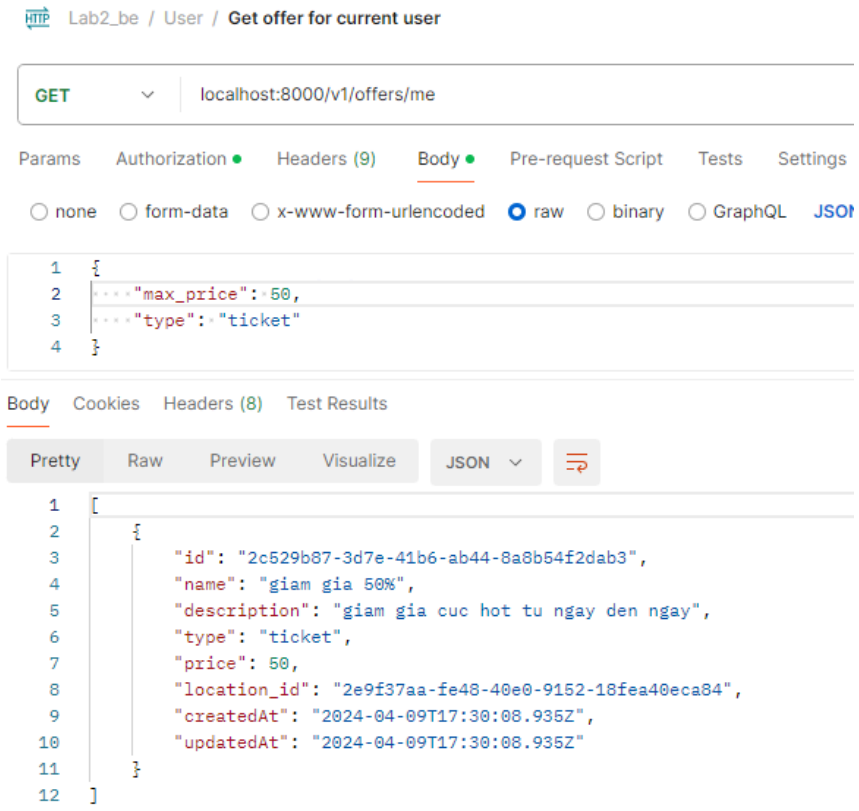
Получить locations, соответствующие интересам пользователя



Получить предложения, соответствующие интересам пользователей



Предложения по запросам соответствуют интересам пользователей



Вывод

В рамках 2-й лаборатории мы научились создавать базовый API для приложения, применять шаблонный код и правильно делить структуру проекта, а также через эту лабораторию мы научились эффективно работать в команде.