Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Кафедра информационных систем и программной инженерии

Лабораторная работа №2

по дисциплине "Технологии разработки веб приложений"

ТЕМА РАБОТЫ:

Разработка серверных приложений

Выполнил:

студент гр. ПРИ-120

Парахин К.В.

Принял:

преподаватель кафедры ИСПИ

Шамышев А.А

Владимир 2024 г

Цель работы:

Изучить различные библиотеки и фреймворки создания сервернойчасти веб-приложений.

Задание:

В ходе выполнения работы вам необходимо 3 раза с использованием различных технологий создать серверную часть веб приложения.

Требование к серверной части:

1. Целостность (система должна предназначаться для выполнения какой-либо одной функции);
2. HTTP API интерфейс (доступ ко всем функциям должен быть через API);
3. Подключение к базе данных;
4. Наличие функции входа в систему и разграничения ролей;
5. Наличие функции загрузки и скачивания файлов (например, аватарок пользователей).

Особые требования: вам необходимо реализовать одно и то же приложение 3 раза с использованием различных технологий. Получившиеся системы должны реализовывать одинаковое поведение на различных версиях. При освоении технологии попытайтесь изучить принципы и паттерны построения приложений, которые применяются при её использовании.

Выполнение работы:

Были выбраны 3 технологии: PHP Laravel, Express JS и Django.

Тематика веб-приложения: приложение для абитуриентов ВУЗа. Представляет собой интерфейс, позволяющий новым абитуриентам регистрироваться/авторизоваться в системе, с помощью администратора (отдельная роль в системе) – добавлять список направлений на поступление, с указанием приоритета, а также отслеживать наличие оригинала диплома. Каждое направление также добавляется администраторов, по умолчанию, для каждого абитуриента в списке проставляется нулевой балл. По получению результатов письменных испытаний, администратор может загрузить .csv файл с оценками – и они обновят старые значения по определенному направлению. Каждое направление имеет свои настройки, на основе которых по финализации результатов – рассчитываются списки поступивших  
Свои результаты абитуриенты могут отслеживать в своем личном кабинете, все действия по изменению их контента производят администраторы в своих личных кабинетах, соответственно.

До разработки клиентских приложений было разработано серверное приложение с REST API, при открытых CORS заголовках для каждого из приложений фронта  
  
Список запросов можно посмотреть в рамках данного окна Postman, все данные приходят через HTTP POST запросы, авторизация происходит с помощью токена, генерирующегося на сервере при регистрации/авторизации пользователя (и сбрасывающегося после его logout действия, соответственно)

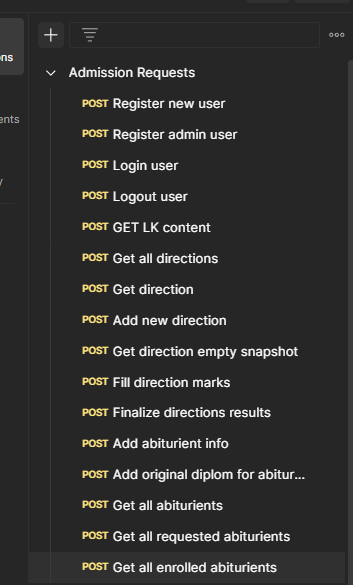


Рисунок 1. Визуализация запросов к API сервера

Собственно, на каждой из 3 технологий (PHP Laravel, Express JS и Django) были реализованы одинаковые методы с теми же самими эндпоинтами.

Каждый сервер использует одну и ту же Postgre SQL базу данных admission\_db, состоящую из 3 таблиц:

BEGIN;

CREATE TYPE admission\_status AS ENUM (

    'request\_in\_progress',

    'enrolled',

    'enrolled\_on\_other',

    'recommended\_on\_other',

    'failed');

CREATE TABLE abiturients(

    id integer GENERATED BY DEFAULT AS IDENTITY,

    email text NOT NULL,

    password text NOT NULL,

    token text NOT NULL,

    is\_admin BOOLEAN DEFAULT FALSE,

    first\_name text NOT NULL,

    second\_name text NOT NULL,

    has\_diplom\_original boolean NOT NULL DEFAULT FALSE,

    is\_requested boolean NOT NULL DEFAULT FALSE,

    is\_enrolled boolean NOT NULL DEFAULT FALSE,

    created\_at VARCHAR(100) NOT NULL DEFAULT '2024-09-15 13:02:56',

    updated\_at VARCHAR(100) NOT NULL DEFAULT '2024-09-15 13:02:56',

    CONSTRAINT pk\_abiturients PRIMARY KEY (id));

CREATE TABLE directions(

    id integer GENERATED BY DEFAULT AS IDENTITY,

    caption text NOT NULL,

    budget\_places\_number INT NOT NULL,

    min\_ball INT NOT NULL DEFAULT 50,

    is\_filled boolean NOT NULL DEFAULT FALSE,

    is\_finalized boolean NOT NULL DEFAULT FALSE,

    created\_at VARCHAR(100) NOT NULL DEFAULT '2024-09-15 13:02:56',

    updated\_at VARCHAR(100) NOT NULL DEFAULT '2024-09-15 13:02:56',

    CONSTRAINT pk\_directions PRIMARY KEY (id));

CREATE TABLE abiturient\_direction\_links(

    id integer GENERATED BY DEFAULT AS IDENTITY,

    abiturient\_id integer NOT NULL,

    direction\_id integer NOT NULL,

    place integer NOT NULL DEFAULT 0,

    mark integer NOT NULL DEFAULT 0,

    admission\_status admission\_status NOT NULL DEFAULT 'request\_in\_progress',

    prioritet\_number integer NOT NULL DEFAULT 1,

    has\_original\_diplom boolean NOT NULL DEFAULT FALSE,

    created\_at VARCHAR(100) NOT NULL DEFAULT '2024-09-15 13:02:56',

    updated\_at VARCHAR(100) NOT NULL DEFAULT '2024-09-15 13:02:56',

    CONSTRAINT pk\_abiturient\_directions\_link PRIMARY KEY (id),

    CONSTRAINT fk\_abiturient\_directions\_link\_abiturients\_abiturient\_id

        FOREIGN KEY (abiturient\_id) REFERENCES abiturients (id)

            ON DELETE CASCADE,

    CONSTRAINT fk\_abiturient\_directions\_link\_directions\_direction\_id

        FOREIGN KEY (direction\_id) REFERENCES directions (id)

            ON DELETE CASCADE);

CREATE OR REPLACE FUNCTION check\_duplicate\_abiturient\_direction()

RETURNS TRIGGER AS $$ BEGIN

IF EXISTS ( SELECT 1 FROM abiturient\_directions\_link

WHERE abiturient\_id = NEW.abiturient\_id AND direction\_id = NEW.direction\_id )

THEN RAISE EXCEPTION

    'Duplicate entry for abiturient\_id % and direction\_id %',

    NEW.abiturient\_id,

    NEW.direction\_id;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent\_duplicate\_abiturient\_direction

BEFORE INSERT ON abiturient\_direction\_links

FOR EACH ROW

EXECUTE FUNCTION check\_duplicate\_abiturient\_direction();

CREATE OR REPLACE FUNCTION restrict\_updates()

RETURNS TRIGGER AS $$ BEGIN

IF NEW.abiturient\_id != OLD.abiturient\_id

    OR NEW.direction\_id != OLD.direction\_id

    OR NEW.place != OLD.place

    OR NEW.has\_original\_diplom != OLD.has\_original\_diplom

THEN RAISE EXCEPTION 'Only mark, admission\_status, and prioritet\_number fields can be updated';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER restrict\_fields\_update

BEFORE UPDATE ON abiturient\_direction\_links

FOR EACH ROW

EXECUTE FUNCTION restrict\_updates();

CREATE OR REPLACE FUNCTION update\_has\_original\_diplom()

RETURNS TRIGGER AS $$ BEGIN

IF NEW.has\_diplom\_original != OLD.has\_diplom\_original

THEN

UPDATE abiturient\_direction\_links

    SET has\_original\_diplom = NEW.has\_diplom\_original,

        updated\_at = CURRENT\_TIMESTAMP

    WHERE abiturient\_id = NEW.id;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger\_update\_has\_original\_diplom

AFTER UPDATE OF has\_diplom\_original

ON abiturients

FOR EACH ROW

EXECUTE FUNCTION update\_has\_original\_diplom();

ALTER TABLE abiturient\_direction\_links RENAME COLUMN has\_original\_diplom TO has\_diplom\_original;

COMMIT;

Сервис на Laravel для быстро доступа к кэшированной информации о пользователе так же использует Redis – в нем он хранит по ключам (идентификаторам пользователя) информацию о его фикстуре в json – формате  
  
Пример:

"content": {

        "first\_name": "Кирилл",

        "second\_name": "Парахин",

        "email": "parahinkv@gmail.com",

        "has\_diplom\_original": true,

        "is\_enrolled": false,

        "directions\_links": [

            {

                "direction\_id": 1,

                "direction\_caption": "PRI",

                "place": 0,

                "mark": 0,

                "admission\_status": "request\_in\_progress",

                "priotitet\_number": 1

            },

            {

                "direction\_id": 2,

                "direction\_caption": "IST",

                "place": 0,

                "mark": 0,

                "admission\_status": "request\_in\_progress",

                "priotitet\_number": 2

            },

            {

                "direction\_id": 3,

                "direction\_caption": "VT",

                "place": 0,

                "mark": 0,

                "admission\_status": "request\_in\_progress",

                "priotitet\_number": 3

            }

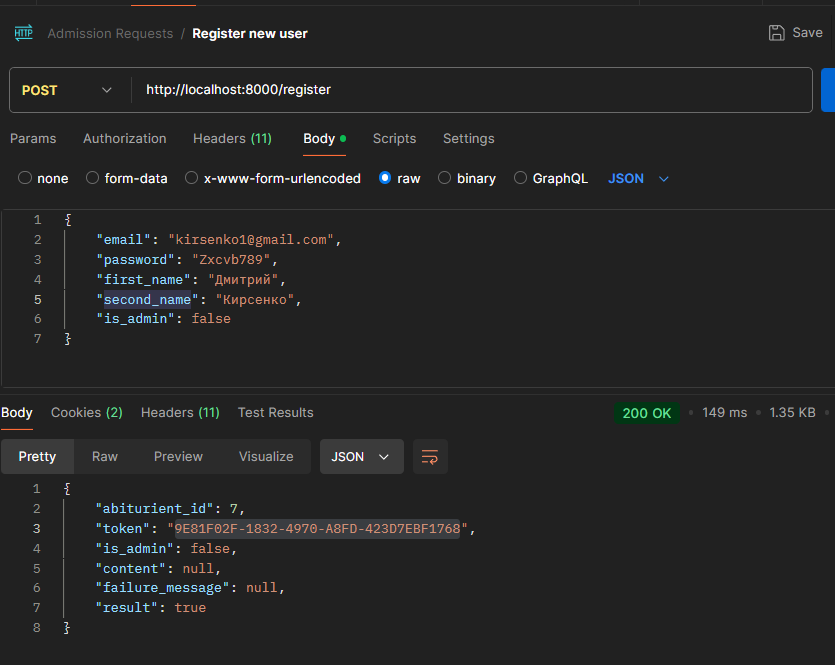
        ]

    },

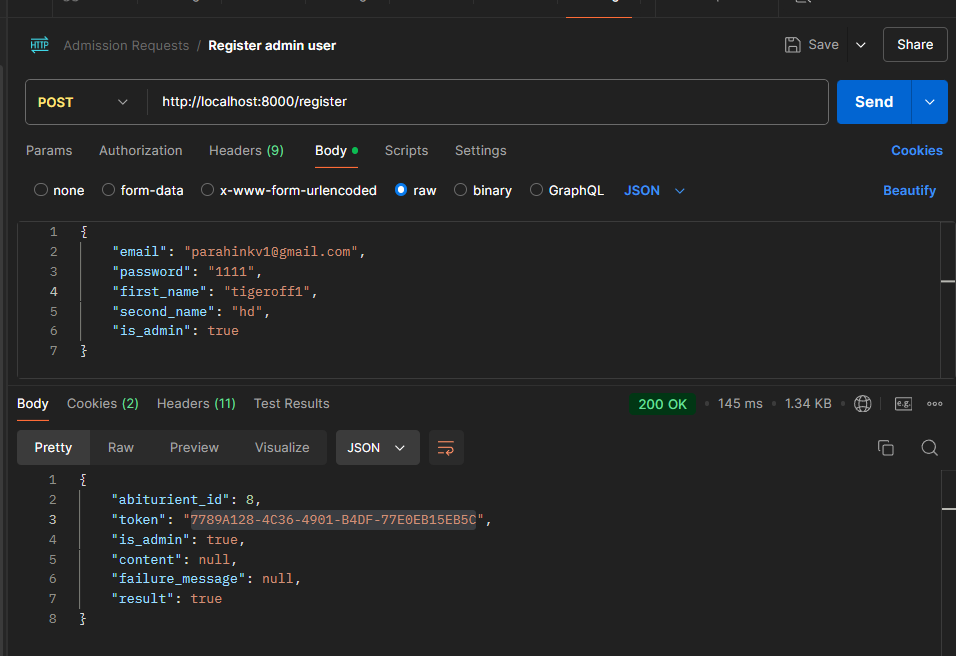
Стоит отметить, что предусматривается 2 роли: обычный пользователь (который может только что то просматривать) и администратор (который обладает правами редактирования контента направлений, абитуриентов и другими функциями) – роль в хранилище определяется флагом is admin – которая указывается пока просто напрямую при регистрации.

Далее протестируем созданное API

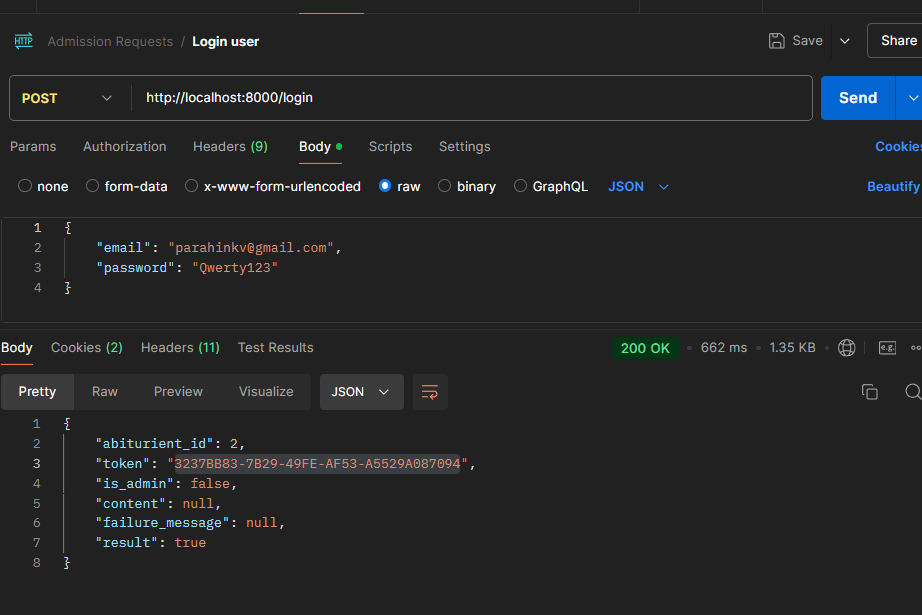
Метод регистрации пользователя



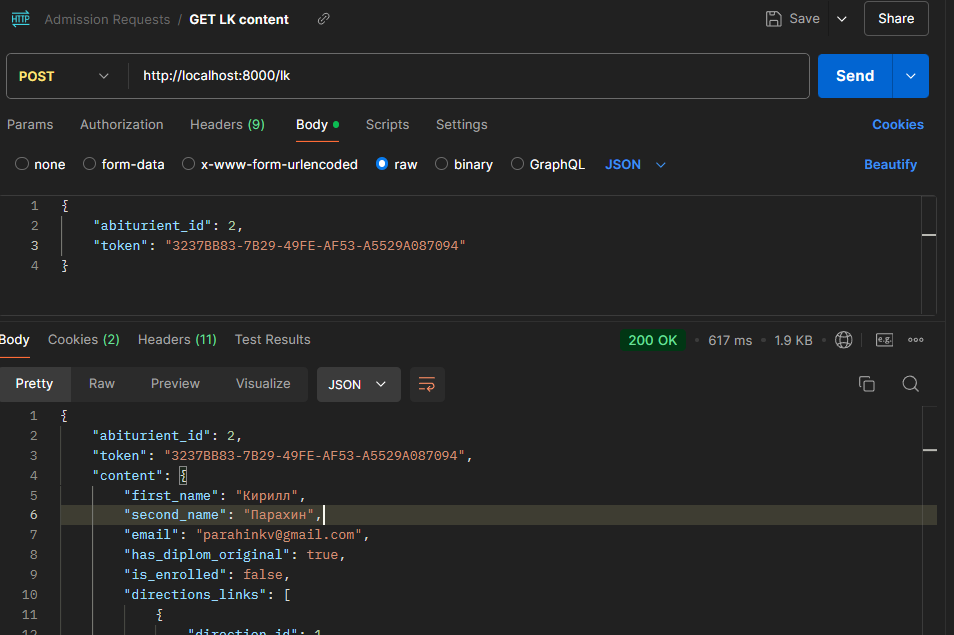
Метод регистрации администратора



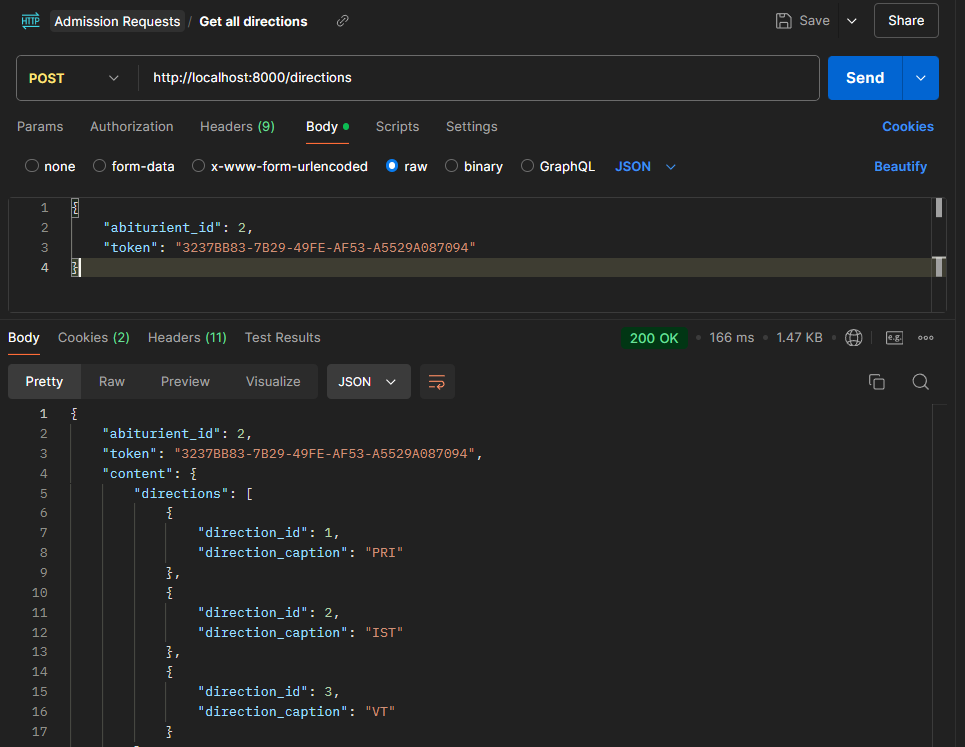
Метод авторизации



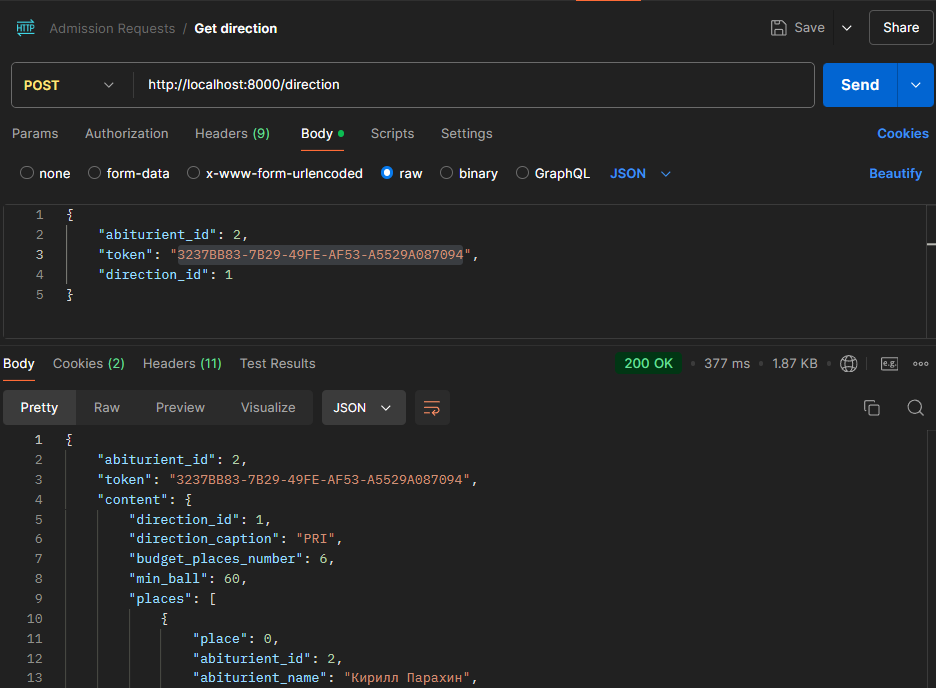
Метод просмотра личного кабинета



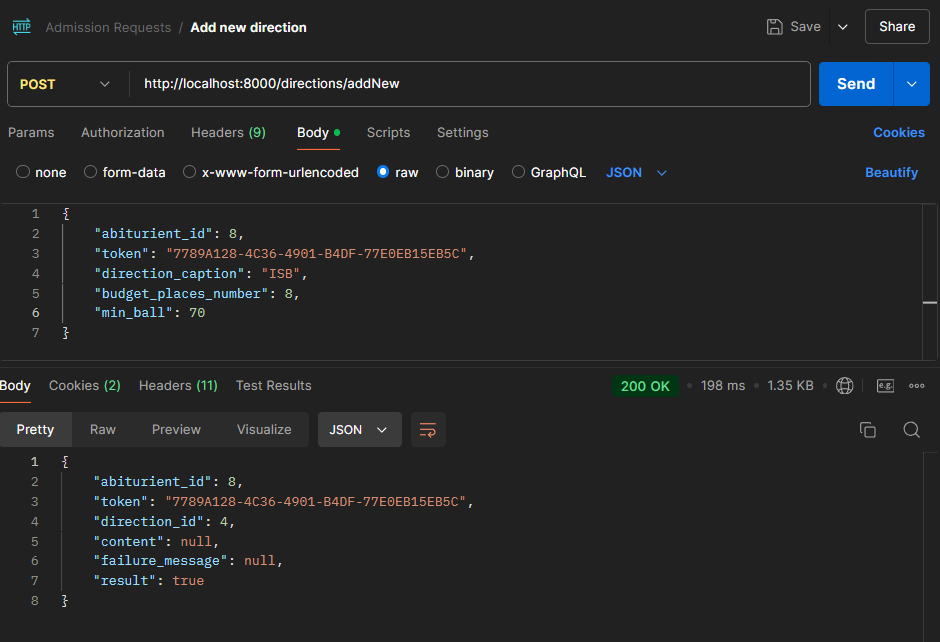
Метод получения списка всех направлений



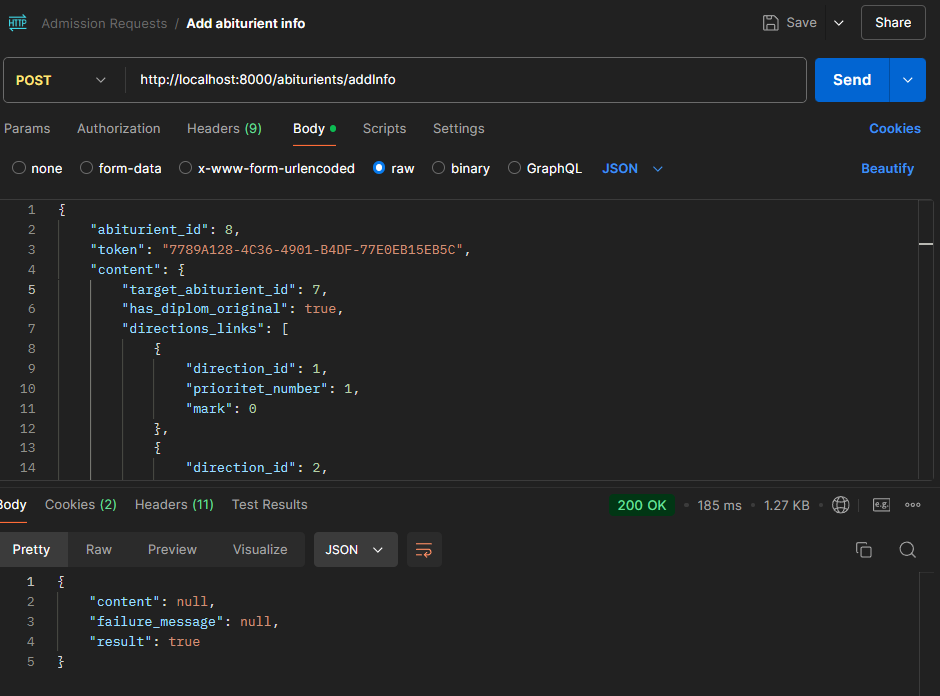
Метод получения страницы направления



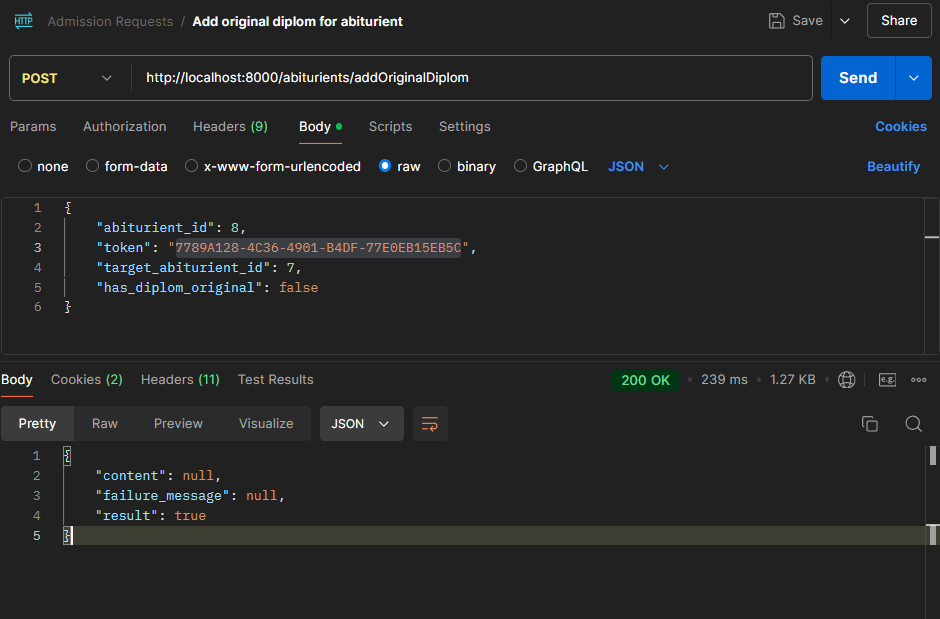
Метод добавления нового направления



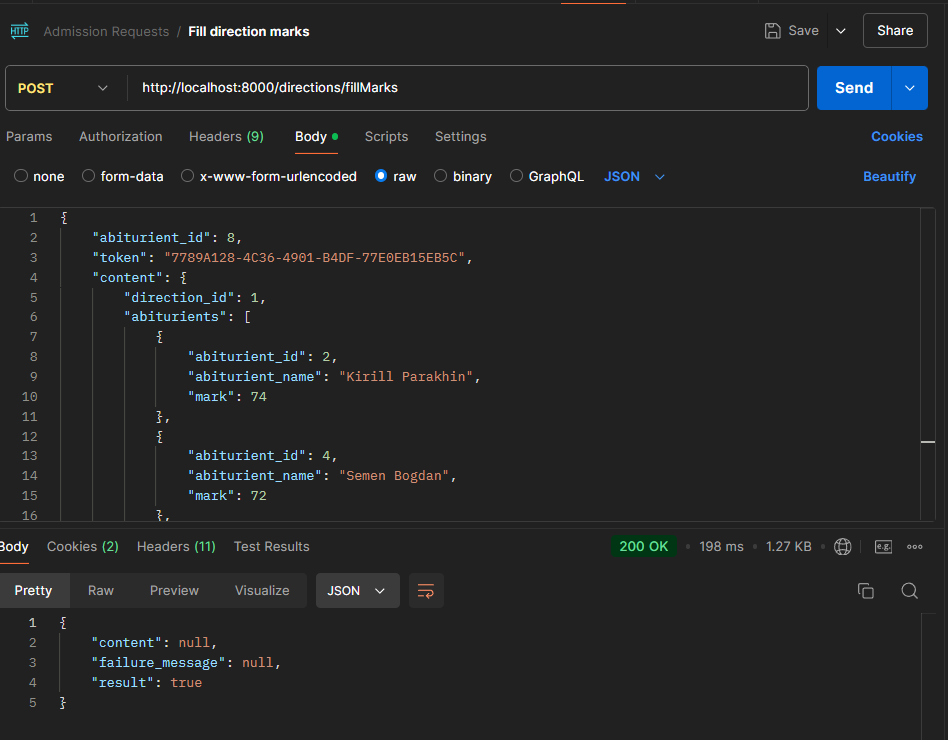
Метод добавления информации абитуриента (и регистрации его направлений)



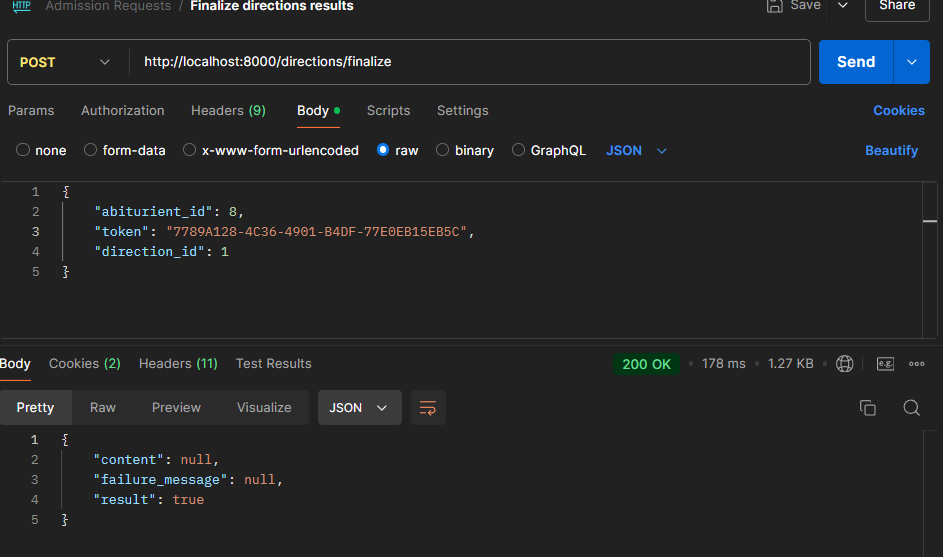
Метод изменения оригинала диплома



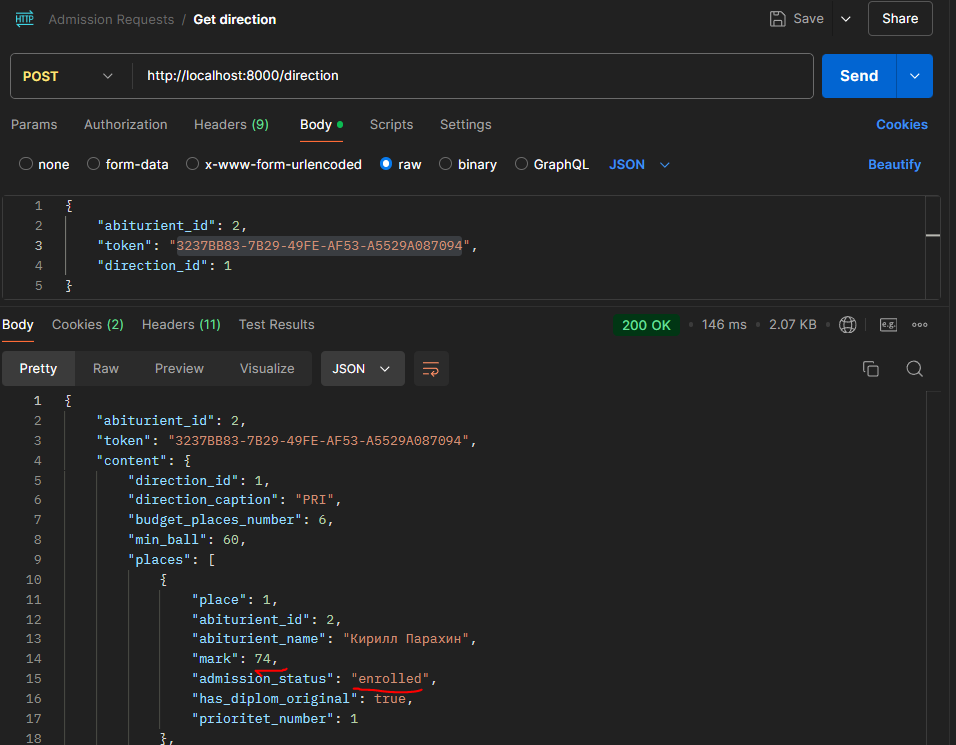
Метод заполнения оценок по направлению

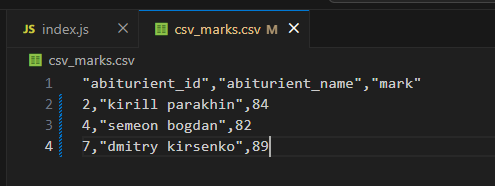


Метод финализации направления



После этого собственно сделав снова запрос к направлению можно заметить, что баллы появились – и у зачисленных студентов сменился статус с «request\_in\_progress» в «enrolled»



Есть еще методы получения списка всех абитуриентов и зачисленных – но их я показывать дополнительно не буду  
  
Также метод с заполнением оценок на express js (в отличие от других технологий) был сделан с помощью HTTP POST запроса с файлом – файл требуется в формате .csv  


Через Postman по каким то причинам его протестить не получилось, поэтому он тестился через запрос к curl:

curl -X POST http://localhost:3030/directions/fillMarks \

     -F 'csv=@./csv\_marks.csv' \

     -F 'abiturient\_id=1' \

     -F 'token=your\_token' \

     -F 'direction\_id=2'



Плюсы и минусы технологий из того что я успел понять:

Плюсы технологий:

1. Laravel  
   Настраиваемый middleware, встроенный удобный Eloquent ORM для удобной работы с БД, понятный синтаксис
2. Express JS  
   Быстрее всего получилось настроить и запустить (установив минимум пакетов), наличие ORM, легковесен, не требует много файлов и входных настроек, хорошо поддерживает асинхронность
3. Django  
   Имеет хорошую масштабируемую структуру, встроенный ORM, много документации и примеров

Минусы технологий:

1. Laravel

Имеются некоторые сложности в первоначальной миграции данных, создает в БД доп таблицы для sessions. Не очень нравится, что маппинг имен метод контроллера и эндпоинтов идет в отдельном файле. Плохо работает асинхронность, в базовом варианте пришлось обойтись вовсе без нее.

1. Express JS

Несмотря на быструю настройку и запуск – структура слишком упрощена и при росте проекта ее будет сложно масштабировать

1. Django

Оказалось сложнее всего запустить из-за создания доп. виртуального окружения, в которое дополнительно нужно было устанавливать пакеты. Предположу, что так как используется python – код будет работать медленнее чем другой код (но зато поддержана асинхронность).

Выводы по использованию технологий:

Для несложных веб - приложений я бы использовал Express JS. Так как опыта больше у меня было на Laravel – то я бы выбрал его для более глубокого изучения, так как он выглядит более продуманным, масштабируемым и гибким.  
Django же показался чуть переусложненным в работе для не Python разработчика (банально проблемы с запуском проекта на другом компьютере возникла).

Вывод:

В результате выполнения работы были освоены принципы создания серверной части веб-приложений с использованием 3 различных технологий.