МИНИСТЕРСТВО НАУКИ И ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Лабораторная работа №03

По дисциплине «Web программирование»

Создание доменной модели

Выполнила студентка группы №M33081

Ахмедова Лейла Таги кызы

Проверил

Приискалов Роман Андреевич

САНКТ-ПЕТЕРБУРГ

2022

Подключаем в качестве СУБД Heroku Postgres

```
heroku addons:create heroku-postgresql:hobby-dev
```

```
PS D:\Lx2\StudentHelper\student-helper> heroku addons:create heroku-postgresql:ho
by-dev
Creating heroku-postgresql:hobby-dev on ⬢ stdnt-frm... free
Database has been created and is available
 ! This database is empty. If upgrading, you can transfer
 ! data from another database with pg:copy
Created postgresql-deep-93548 as HEROKU_POSTGRESQL_BRONZE_URL
Use heroku addons:docs heroku-postgresql to view documentation
PS D:\Lx2\StudentHelper\student-helper>
```

Выбираем ORM Prisma

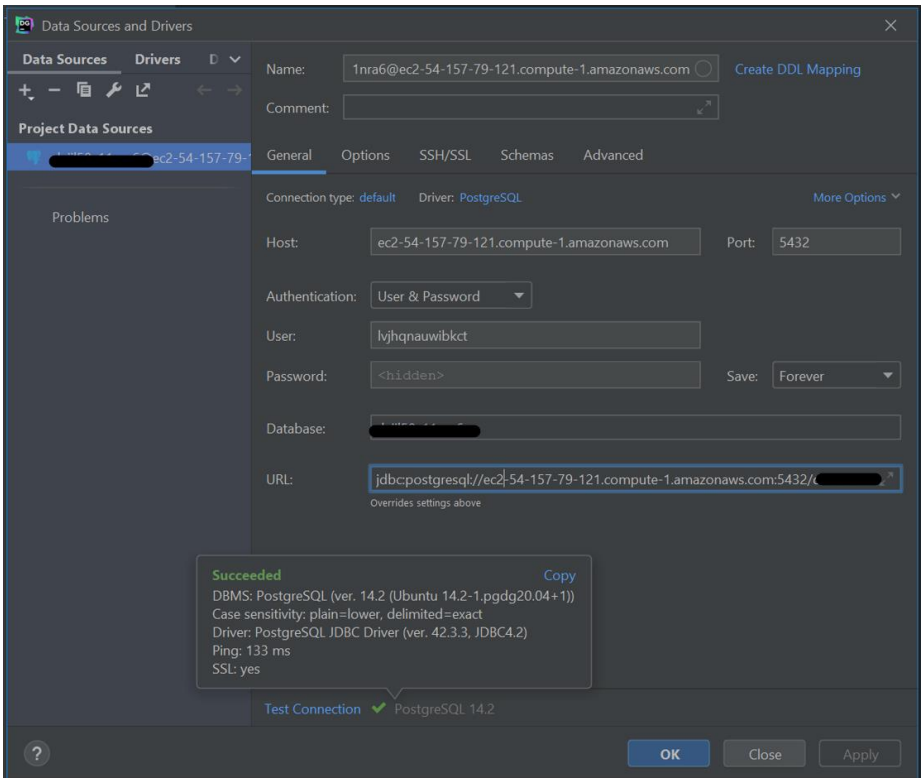
Создаем начальную настройку Prisma

```
$ npx prisma init
```

Эта команда создает каталог **prisma** с файлом

- **schema.prisma** – содержит подключение к базе данных и содержит схему бд
- **.env** – используется для хранения учетных данных бд в группе переменных среды

Проверяем соединение к бд, использую DataGrip



Далее в файл добавляем две модели таблицы User и Post

```
generator client {
  provider = "prisma-client-js"
}
datasource db {
  provider = "postgresql"
  url               = env("DATABASE_URL")
  shadowDatabaseUrl = env("SHADOW_DATABASE_URL")
}
model User {
  id    Int     @default(autoincrement()) @id
  email String  @unique
  name  String?
  posts Post[]
  isLoggedIn Boolean @default(false)
}

model Post {
  id        Int      @default(autoincrement()) @id
  title     String
  content   String?
  published Boolean? @default(false)
  author    User?    @relation(fields: [authorId], references: [id])
  authorId  Int?
}
```

После изменений в бд выполняем миграцию

```
$ npx prisma migrate dev --name init
```

Эта команда генерирует файлы SQL и напрямую запускает их в бд

```
migrations/
  └─ 20220402161520_init/
    └─ migration.sql

Your database is now in sync with your schema.
```

! Перед этим создаем теневую бд. При создании новой миграции Prisma использует теневую бд для обнаружения смещения схемы (т. е. не было внесено никаких изменений вручную).

После миграции устанавливаем Prisma Client

```
$ npm install @prisma/client
```

Теперь можем отправлять запросы к бд с помощью Prisma Client.

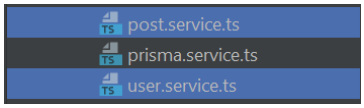
Внутри src создаю новый файл prisma.service.ts

Нужен для созданий экземпляра PrismaClient и подключения к нашей бд

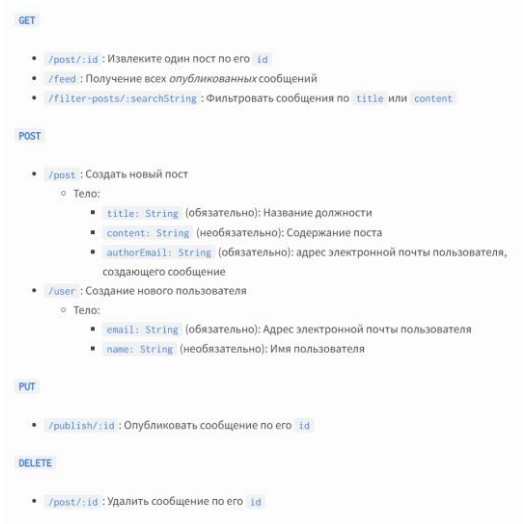
```
import { INestApplication, Injectable, OnModuleInit } from '@nestjs/common';
import { PrismaClient } from '@prisma/client';

@Injectable()
export class PrismaService extends PrismaClient implements OnModuleInit {
  async onModuleInit() {
    await this.$connect();
  }

  async enableShutdownHooks(app: INestApplication) {
    this.$on('beforeExit', async () => {
      await app.close();
    });
  }
}
```

Теперь можем создать службы для вызовов бд для наших двух моделей.

Создаем два файла в папке src: **user.service.ts** и **post.service.ts**



Теперь добавим дополнительные маршруты в существующий класс **app.controller.ts**



`GET`

- `/post/:id` : Извлеките один пост по его `id`
- `/feed` : Получение всех *опубликованных* сообщений
- `/filter-posts/:searchString` : Фильтровать сообщения по `title` или `content`

`POST`

- `/post` : Создать новый пост
  - Тело:
    - `title: String` (обязательно): Название должности
    - `content: String` (необязательно): Содержание поста
    - `authorEmail: String` (обязательно): адрес электронной почты пользователя, создающего сообщение
- `/user` : Создание нового пользователя
  - Тело:
    - `email: String` (обязательно): Адрес электронной почты пользователя
    - `name: String` (необязательно): Имя пользователя

`PUT`

- `/publish/:id` : Опубликовать сообщение по его `id`

`DELETE`

- `/post/:id` : Удалить сообщение по его `id`

Заливаем в Heroku, проверяем, что в DataGrip появились наши таблицы и смотрим их визуализацию