

Миграция базы данных в проекте Nodejs с помощью ORM Prisma

Вихляев Дмитрий Романович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В данной статье рассматривается подключение инструмента Prisma к Nodejs проекту для реализации миграции базы данных. Для реализации используется npm пакет Prisma клиент и CLI, язык программирования JavaScript и СУБД Postgresql. В результате исследования будет описан метод взаимодействия ORM Prisma с приложением для взаимодействия с базой данных.

Ключевые слова: ORM, Prisma, docker, Nodejs.

Database migration in a Nodejs project using ORM Prisma

Vikhlyayev Dmitry Romanovich

Sholom-Aleichem Priamursky State University

Student

Abstract

This article discusses connecting the Prisma tool to a Nodejs project to implement database migration. The npm package Prisma client and CLI, the JavaScript programming language and the Postgresql database management system are used for implementation. As a result of the study, the ORM Prisma interaction method with the database interaction application will be described.

Keywords: ORM, Prisma, docker, Nodejs.

1 Введение

1.1 Актуальность

Современный инструмент Prisma применяется для работы с базами данных, который обеспечивает высокую продуктивность разработки за счёт автоматической генерации типобезопасного клиентского API, простоты управления схемой базы данных и удобного механизма миграций. В условиях растущей сложности приложений актуальность использования Prisma обусловлена необходимостью минимизации ошибок при взаимодействии с базой данных, а также упрощением процессов разработки и сопровождения. Особенностью Prisma является использование декларативного подхода для описания структуры данных через файл `schema.prisma`. Это позволяет разработчикам сосредоточиться на моделировании данных, избегая ручного написания SQL-кода. На основе этого описания Prisma автоматически генерирует клиент, который предоставляет строго типизированные методы

для выполнения операций с базой данных. Такая типизация помогает выявлять ошибки на этапе компиляции, улучшая надёжность кода.

1.2 Обзор исследований

И.Д.Грязных, И.В.Ананченко, А.В.Гайков использовали Prisma ORM для упрощения работы с базой данных MySQL в бэкенд-части своей системы. [1]. И.В.Ананченко, В.И.Семенов Разработали программное обеспечение для автоматизации ведения репозитория университета с помощью инструментов NestJS, MySQL и Prisma ORM [2]. А.А.Мишин, А.А.Гриценко О.И.Белозеров описали разработку программного модуля для преобразования данных СУБД [3]. В.Л.Шмелев, Е.Ю.Воронкин создали программное обеспечение для метеорологической службы [4]. Ю.А.Королева, В.О.Маслова, В.К.Козлов представили концепцию миграции данных между реляционными и нереляционными системами БД [5].

1.3 Цель исследования

Цель исследования – описать метод взаимодействия приложения Nodejs с базой данных через ORM Prisma.

2 Материалы и методы

Для реализации используется интерпретатор Javascript Nodejs и установщик пакетов npm. В качестве СУБД используется Postgresql. Программа docker для создания образа через Dockerfile и контейнера через docker-compose.yml.

3 Результаты и обсуждения

Prisma – это объектно-реляционный маппер (ORM) для Node.js, который предоставляет простой и удобный способ взаимодействия с базой данных. Он поддерживает типобезопасность, статическую проверку типов и автоматическую генерацию клиентского API. Prisma позволяет разработчику сосредоточиться на бизнес-логике, упрощая задачи, связанные с управлением данными, такими как создание, чтение, обновление и удаление записей. Для работы с Prisma необходимо выполнить несколько шагов, включая установку, настройку и использование в проекте. Каждый из этапов важен для корректного функционирования Prisma в приложении.

Первым шагом является установка необходимых пакетов. В проект добавляются два пакета: prisma для управления схемой и миграциями базы данных, а также @prisma/client для взаимодействия с базой данных. Команды для установки «npm i -D prisma». Пакет prisma устанавливается как зависимость для разработки. Это связано с тем, что пакет prisma используется исключительно для задач, связанных с управлением схемой базы данных и миграциями. Он не нужен в рабочем окружении (production). В рабочей среде приложение использует только сгенерированный клиент @prisma/client. Этот пакет содержит всё необходимое для выполнения запросов к базе данных.

Команда «`npx prisma init`» используется для инициализации проекта с Prisma. Она создаёт базовую структуру и конфигурационные файлы, необходимые для работы. Это создаёт директорию `prisma` с файлом конфигурации `schema.prisma`, где описываются модели данных. Созданный файл `schema.prisma` содержит информацию о подключении к базе данных и модели данных. В данном примере используется PostgreSQL, и строка подключения берётся из переменной окружения «`DATABASE_URL`». Для запуска `nodejs` сервера в `docker` контейнере, потребуется добавить директиву «`binaryTargets`» в файл `schema.prisma`, которая используется для настройки генерации бинарных файлов Prisma Client, оптимизированных для выполнения на определённых платформах. «`native`» указывает, что Prisma Client будет скомпилирован для платформы, на которой выполняется команда `npx prisma generate`. «`debian-openssl-3.0.x`»: Указывает, что необходимо сгенерировать клиент для окружений, совместимых с Debian и OpenSSL версии 3.0.x. (рис.1,2).

```
model users {
  id      Int      @id @default(autoincrement())
  username String   @db.VarChar(50)
  password String   @db.VarChar(100)
  role    String?  @db.VarChar(20)
  email   String?
}

model stations {
  id      Int      @id @default(autoincrement())
  name    String    @db.VarChar(255)
  latitude Float
  longitude Float
  measurements measurements[]
}

model measurements {
  id      Int      @id @default(autoincrement())
  date    DateTime @db.Date
  temperature Float
  humidity Float
  wind_speed Float
  station_id Int
  station stations @relation(fields: [station_id], references: [id])
}
```

Рис. 1. Схема базы данных в Prisma

```
generator client {
  provider = "prisma-client-js"
  binaryTargets = ["native", "debian-openssl-3.0.x"]
}

datasource db {
  provider = "postgresql"
  url       = env("DATABASE_URL")
}
```

Рис. 2. Конфигурация Prisma

Миграции позволяют синхронизировать базу данных с описанием моделей в `schema.prisma`. После внесения изменений в файл схемы необходимо создать новую миграцию. Команда «`npx prisma migrate dev --`

name <название миграции>» создаст sql файл с описанными таблицами и полями в схеме. Когда будет необходимо добавить новые таблицы или поля, изменить тип данных или индексы, переделывается сама схема, а миграция добавит новый sql в директории миграции, с изменениями относительно предыдущего. Например, при добавлении нового поля, новый sql файл будет содержать только одну команду «ALTER TABLE ...» (рис.3).

```
-- CreateTable
CREATE TABLE "users" (
  "id" SERIAL NOT NULL,
  "username" VARCHAR(50) NOT NULL,
  "password" VARCHAR(100) NOT NULL,
  "role" VARCHAR(20),
  "email" TEXT,

  CONSTRAINT "users_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "stations" (
  "id" SERIAL NOT NULL,
  "name" VARCHAR(255) NOT NULL,
  "latitude" DOUBLE PRECISION NOT NULL,
  "longitude" DOUBLE PRECISION NOT NULL,
  CONSTRAINT "stations_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "measurements" (
  "id" SERIAL NOT NULL,
  "date" DATE NOT NULL,
  "temperature" DOUBLE PRECISION NOT NULL,
  "humidity" DOUBLE PRECISION NOT NULL,
  "wind_speed" DOUBLE PRECISION NOT NULL,
  "station_id" INTEGER NOT NULL,
  CONSTRAINT "measurements_pkey" PRIMARY KEY ("id"),
  CONSTRAINT "measurements_station_id_fkey" FOREIGN KEY ("station_id") REFERENCES "stations" ("id") ON DELETE RESTRICT ON UPDATE CASCADE;
);

-- AddForeignKey
ALTER TABLE "measurements" ADD CONSTRAINT "measurements_station_id_fkey"
FOREIGN KEY ("station_id") REFERENCES "stations" ("id") ON DELETE RESTRICT ON UPDATE CASCADE;
```

Рис. 3. Миграция базы данных

Для взаимодействия с базой данных Prisma генерирует клиентский API на основе описания моделей в schema.prisma «npx prisma generate». Этот процесс создаёт тип безопасный клиент, который можно использовать в коде приложения. Предыдущая команда «migrate dev» выполняет сначала миграцию, а затем генерирует клиента.

После генерации, клиент становится доступным для использования в приложении через пакет @prisma/client. Для тестирования и начального заполнения базы данных можно использовать механизм seed (посева данных). Этот процесс автоматизирует добавление тестовых данных в базу. Обычно файл seed.js или seed.ts создаётся в директории проекта (рис.4). В этом файле используется Prisma Client для добавления данных в базу. В файле package.json добавляется команда для выполнения скрипта (рис.5) Для выполнения скрипта используется команда «npx prisma db seed».

```

const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

async function main() {
  console.log("Seeding database...");
  // Заполнение таблицы users
  await prisma.users.createMany({
    data: [
      { username: 'admin', password: 'admin12', role: 'Admin' },
      { username: 'prioritet', password: 'prioritet13', role: 'Priority' },
      { username: 'guest', password: 'guest14', role: 'guest' },
    ],
  });
  // Заполнение таблицы stations
  await prisma.stations.createMany({
    data: [
      { name: 'Station Alpha', latitude: 40.712776, longitude: -74.005974 },
      { name: 'Station Beta', latitude: 34.052235, longitude: -118.243683 },
      { name: 'Station Gamma', latitude: 41.878113, longitude: -87.629799 },
      { name: 'Station Delta', latitude: 29.760427, longitude: -95.369804 },
    ],
  });
  // Заполнение таблицы measurements
  await prisma.measurements.createMany({
    data: [
      { date: new Date('2025-01-08'), temperature: 15.2, humidity: 60.1, wind_speed: 5.3, station_id: 1 },
      { date: new Date('2025-01-09'), temperature: 16.5, humidity: 58.3, wind_speed: 4.7, station_id: 1 },
      { date: new Date('2025-01-08'), temperature: 20.4, humidity: 45.7, wind_speed: 3.1, station_id: 2 },
      { date: new Date('2025-01-09'), temperature: 21.0, humidity: 44.9, wind_speed: 3.5, station_id: 2 },
      { date: new Date('2025-01-08'), temperature: -5.0, humidity: 80.2, wind_speed: 15.0, station_id: 3 },
      { date: new Date('2025-01-09'), temperature: -4.8, humidity: 79.9, wind_speed: 14.5, station_id: 3 },
      { date: new Date('2025-01-08'), temperature: 25.1, humidity: 50.3, wind_speed: 7.8, station_id: 4 },
      { date: new Date('2025-01-09'), temperature: 26.0, humidity: 49.0, wind_speed: 8.2, station_id: 4 },
    ],
  });
  console.log("Seeding completed.");
}

main()
  .catch((e) => {
    console.error(e);
    process.exit(1);
  })
  .finally(async () => {
    await prisma.$disconnect();
  });

```

Рис. 4. Содержимое seed.js

```

{
  "prisma": {
    "seed": "node prisma/seed.js"
  }
}

```

Рис. 5. Команда запуска package.json

Модуль «@prisma/client» предоставляет типобезопасный способ выполнения операций с данными, таких как создание, чтение, обновление и удаление (CRUD). Запросы Prisma Client имеют стандартную структуру «prisma.<модель>.<метод>(параметры)». Модель соответствует имени модели из schema.prisma. Например, user, post. Метод – это операция, которую нужно выполнить (например, create, findMany, update). Параметры представляют объект, задающий условия выполнения запроса (например, where, data, select) (рис.6).

```

const getUser = async (param) => {
  try {
    const result = await prisma.users.findMany({ select: { username:true, password:true,role:true } });
    return result;
  } catch (err) {
    console.log(err.message);
    res.status(500).json({ message: err });
  }
};

const updateUser = async (req, res) => {
  const { id } = req.params;
  const { username, password, role } = req.body;
  try {
    const updatedUser = await prisma.users.update({
      where: { id: Number(id) },
      data: { username, password, role },
    });
    res.json(updatedUser);
  } catch (error) {
    if (error.code === 'P2025') {
      res.status(404).json({ error: 'User not found' });
    } else {
      res.status(500).json({ error: error.message });
    }
  }
};

const deleteUser = async (req, res) => {
  const { id } = req.params;
  try {
    await prisma.users.delete({ where: { id: Number(id) } });
    res.status(204).send();
  } catch (error) {
    if (error.code === 'P2025') {
      res.status(404).json({ error: 'User not found' });
    } else {
      res.status(500).json({ error: error.message });
    }
  }
};

```

Рис. 6. Методы SELECT, UPDATE, DELETE

Далее приводится пример запуска приложения в контейнере docker. Чтобы prisma правильно сгенерировала клиента для операционной системы Debian нужно добавить в схему директиву «binaryTargets» представленной на рисунке 2. Через декларативное описание docker-compose (рис.7), создаётся контейнер на основе образа построенного из инструкций Dockerfile (рис.8). Команда «npx prisma migrate deploy» выполняет производственную миграцию, на основе уже имеющихся изменений. Использование «npx prisma generate» нужно для регенерации клиента при переходе с Windows на Linux.

```

node:
  build: "./app"
  environment:
    DATABASE_URL: "postgresql://postgres:pass@postgres:5432/meteo?schema=public"
  ports:
    - "3000:3000"
  command: sh -c "npx prisma migrate deploy && npx prisma generate && npx prisma db seed && npm start"

```

Рис. 7. Фрагмент docker-compose

```

FROM node:20.16
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .

```

Рис. 8. Содержимое Dockerfile

4 Выводы

В результате исследования был описан процесс создания миграций с помощью ORM Prisma. Представлены основные настройки и методы взаимодействия с программной средой. Приведён пример обёртки программы в docker контейнер с последующей миграцией и заполнением данных.

Библиографический список

1. Грязных И.Д., Ананченко И.В., Гайков А.В. Разработка программного обеспечения для автоматизации ведения балльно-рейтинговой системы вуза // Современные наукоемкие технологии. 2023. № 6. С. 20-26.
2. Ананченко И.В., Семенов В.И. Разработка программного обеспечения для автоматизации ведения репозитория университета // В сборнике: Global science. Сборник статей II Международной научно-практической конференции. Пенза, 2023. С. 58-63.
3. Мишин А.А., Гриценко А.А., Белозеров О.И. Разработка программного модуля для преобразования данных СУБД // Научно-техническое и экономическое сотрудничество стран АТР в XXI веке. 2022. Т. 1. С. 254-259.
4. Шмелев В.Л., Воронкин Е.Ю. Разработка программного обеспечения для метрологической службы // Интерэкспо Гео-Сибирь. 2023. Т. 6. № 1.
5. Королева Ю.А., Маслова В.О., Козлов В.К. Разработка концепции миграции данных между реляционными и нереляционными системами БД // Программные продукты и системы. 2019. № 1. С. 063-067.