Лабораторная работа 19

ПСКП

ПОИТ-3

**Задание 01**

1. Используйте БД из предыдущей лабораторной работы.

**Задание 02**

1. Разработайте приложение **19-01**, представляющее собой HTTP-сервер, прослушивающий порт ***3000***. Сервер должен обрабатывать запросы, описанные в следующих таблицах. **Для выполнения SQL-запросов используйте пакет prisma**.

|  |  |  |
| --- | --- | --- |
| Метод  запроса | URI | Назначение |
| GET | / | получить статический HTML-файл, назначение файла описывается в задании 03 |
| GET | api/faculties | получить список всех факультетов в json-формате |
| GET | api/pulpits | получить список всех кафедр в json-формате |
| GET | api/subjects | получить список всех учебных дисциплин в json-формате |
| GET | api/teachers | получить список всех преподавателей в json-формате |
| GET | api/auditoriumstypes | получить список всех типов учебных аудиторий в json-формате |
| GET | api/auditoriums | получить список всех учебных аудиторий в json-формате |
| GET | api/faculties/xyz/subjects | xyz – код факультета, получить за один запрос факультет xyz с его дисциплинами через связанные записи. В результате должно возвращаться следующее: |
| GET | api/auditoriumtypes/xyz/  auditoriums | xyz – код типа аудитории, получить аудитории с типом аудитории xyz через связанные записи. В результате должно возвращаться следующее: |
| GET | api/auditoriumsWithComp1 | Получить список всех компьютерных классов первого корпуса |
| GET | api/puplitsWithoutTeachers | Получить список всех кафедр без преподавателей |
| GET | api/pulpitsWithVladimir | Получить список всех кафедр, на которых есть хотя бы один преподаватель с именем Владимир |
| GET | api/auditoriumsSameCount | Получить количество всех аудиторий с одинаковым типом и вместимостью |
| POST | api/faculties+ | добавить новый факультет, данные в json-формате. Должна быть возможность вместе с новым факультетом сразу создавать кафедры, которые к нему относятся. Связанные кафедры должны создаваться лишь в том случае, когда они передаются в запросе. |
| POST | api/pulpits+ | добавить новую кафедру, данные в json-формате. Факультет, к которому относится кафедра передавать в формате «faculty, faculty\_name». В случае, если факультета с указанным кодом faculty не существует, то создавать новый (faculty\_name брать из строки) |
| POST | api/subjects+ | добавить новую учебную дисциплину, данные в json-формате. |
| POST | api/teachers+ | добавить нового преподавателя, данные в json-формате |
| POST | api/auditoriumstypes | добавить новый тип учебной аудитории, данные в json-формате |
| POST | api/auditoriums | добавить новую учебную аудиторию, данные в json-формате |
| PUT | api/faculties | корректировать информацию о факультете, данные в json-формате |
| PUT | api/pulpits | корректировать информацию о кафедре, данные в json-формате |
| PUT | api/subjects | корректировать информацию об учебной дисциплине, данные в json-формате |
| PUT | api/teachers | корректировать информацию о преподавателе, данные в json-формате |
| PUT | api/auditoriumstypes | корректировать информацию о типе учебной аудитории, данные в json-формате |
| PUT | api/auditoriums | корректировать информацию об аудитории, данные в json-формате |
| DELETE | api/faculties/xyz | xyz - код факультета, удалить факультет с кодом xyz |
| DELETE | api/pulpits/xyz | xyz - код кафедры, удалить кафедру с кодом xyz |
| DELETE | api/subjects/xyz | xyz - код дисциплины, удалить дисциплину с кодом xyz |
| DELETE | api/teachers/xyz | xyz - код преподавателя, удалить преподавателя с кодом xyz |
| DELETE | api/auditoriumtypes/xyz | xyz - код типа аудитории, удалить тип аудитории с кодом xyz |
| DELETE | api/auditoriums/xyz | xyz - код аудитории, удалить аудиторию с кодом xyz |

|  |  |
| --- | --- |
| Метод  запроса | Назначение |
| GET | Возвращает HTML-файл, найденные в БД данные или ***сообщение об ошибке в json-формате*** |
| POST | принимает данные в json-формате,  возвращает добавленные в БД данные или ***сообщение об ошибке в json-формате*** |
| PUT | принимает данные в json-формате,  возвращает измененные в БД данные или ***сообщение об ошибке в json-формате*** |
| DELETE | возвращает удаленные в БД данные или ***сообщение об ошибке в json-формате*** |

1. Обязательно проверяйте входящие данные.
2. Для хранения данных приложение **19-01** должно использовать БД **XYZ.** Для соединения с БД сервердолжен применять пул соединений.
3. Для проверки работоспособности приложения, используйте **POSTMAN**.

**Задание 03**

1. Напишите какой-то пример использования Fluent API. Объясните, что это такое и для чего применяется.

**Задание 04**

1. Разработайте HTML-страницу, на которой будет выводиться информация обо всех кафедрах.
2. Список кафедр должен выводиться по 10 записей на странице. Внизу должны отображаться кнопки с номером страницы.
3. Рядом с информацией о каждой кафедре должно дополнительно выводиться количество преподавателей на ней (использовать \_count).

**Задание 05**

1. Добавьте транзакцию, в которой происходит изменение во всех аудиториях вместительности на 100 (использовать инкремент), а потом сразу же изменения откатываются.
2. Продемонстрируйте работу транзакции.

**Задание 06** Ответьте на следующие вопросы

1. Что такое Prisma?

это ORM нового поколения с открытым исходным кодом для Node.js и TypeScript. Она состоит из следующих инструментов:

 Prisma Client: автогенерируемый и типобезопасный клиент базы данных;

 Prisma Migrate: система миграций;

 Prisma Studio: пользовательский интерфейс для просмотра и редактирования данных.

1. Назовите отличия Prisma от других фреймворков ORM?

 написан на Rust;

 реализует паттерн Data Mapper;

 поддерживаются MySQL, PostgreSQL, MSSQL и SQLite (поддержка MongoDB, CockroachDB и PlanetScale в предварительной версии);

 предоставляет типобезопасный API (TypeScript);

 имеет набор инструментов для работы с базами данных (отправка запросов, моделирование, миграции, прототипирование, data seeding, студия для просмотра и изменения);

 способен генерировать определение схемы базы данных и клиентский код на основе структуры базы данных;

 поддерживает различные параметры запросов (фильтрация, сортировка, группировка, пагинация и др.);

1. Какова роль Prisma Client в доступе к данным и управлении ими?

Prisma Client является основным инструментом, который используется для управления данными в приложении, написанном на Prisma.

Prisma Client позволяет разработчикам писать безопасный, эффективный и типизированный код при работе с базами данных, что упрощает разработку и поддержку приложений, особенно в больших проектах.

Prisma Client - это автоматически сгенерированный и типобезопасный конструктор запросов, адаптированный к вашим данным. Самый простой способ начать работу с Prisma Client - следовать инструкциям **[Quickstart](https://www.prisma.io/docs/getting-started/quickstart)**.

1. Можно ли использовать Prisma с базами данных SQL и NoSQL?

Prisma — это инструмент, позволяющий работать с реляционными (PostgreSQL, MySQL, SQL Server, SQLite) и нереляционной (MongoDB) базами данных с помощью JavaScript или TypeScript без использования SQL (хотя такая возможность имеется).Какие языки программирования совместимы с Prisma?

Поддерживаемые языки программирования:

* JavaScript
* TypeScript
* Go (в разработке)

Да.

1. Расскажите, где и как можно использовать ***every-*** "Мне нужен список всехUser, в которых есть только неопубликованные Postзаписи", ***none-*** Мне нужен список всехUser, у которых нет неопубликованных Postзаписей и ***some-*** Мне нужен список всехUser, в которых есть хотя бы одна неопубликованная Postзапись"?
2. Для чего нужен метод ***upsert***?
3. 
4. Как включить регистронезависимый поиск?

Для включения регистронезависимого поиска в Prisma необходимо использовать инструкцию **iLike** при построении запросов к базе данных.

1. 
2. Объясните назначение методов ***$queryRaw***-$queryRaw возвращает фактические записи базы данных. Например, следующий SELECTзапрос возвращает все поля для каждой записи в Userтаблице:

***$executeRaw***.

* $queryRaw для возврата фактических записей (например, с помощью SELECT)
* $executeRaw для возврата количества затронутых строк (например, после UPDATE или DELETE)

1. Какие виды транзакций существуют в Prisma?

Prisma поддерживает следующие виды транзакций:

prisma.$transaction - это транзакция на уровне запросов. Она позволяет группировать несколько запросов в единую транзакцию. Если хотя бы один запрос не будет выполнен успешно, то все изменения будут отменены.

prisma.$executeRaw - это транзакция на уровне SQL. Она позволяет выполнить несколько SQL-запросов в единой транзакции.

prisma.transaction() - это транзакция на уровне моделей данных. Она позволяет группировать несколько операций на модели данных в единую транзакцию. Если хотя бы одна операция не будет выполнена успешно, то все изменения будут отменены.