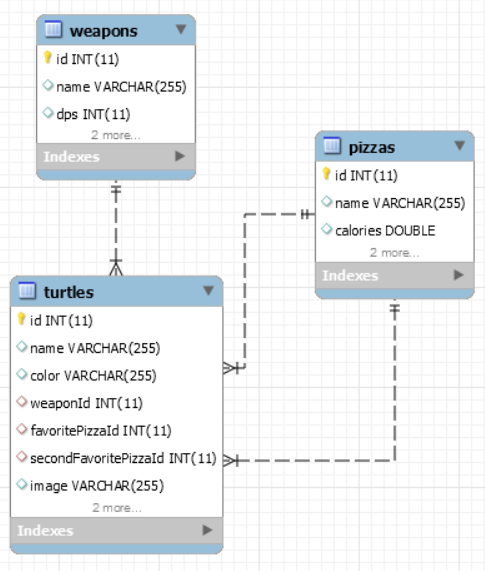
Лабораторная работа 06

СТРWП

**Задание 01**

1. Ознакомьтесь с диаграммой базы данных (БД)



1. Создайте собственный экземпляр БД с именем: **XYZ**, где XYZ – инициалы студента. Можно использовать любую реляционную СУБД, поддерживаемую sequelize/prisma.

**Задание 02**

1. Для взаимодействия в БД используйте пакет **sequelize** или пакет **prisma**.
2. Сперва используйте подход database first.

prisma db pull

1. Затем удалите БД и для повторного ее создания используйте подход code first.

prisma migrate dev

prisma migrate deploy

1. Заполните БД данными.

**Задание 03**

1. Разработайте приложение **06-01**, представляющее собой **EXPRESS**-HTTP-сервер, прослушивающий порт ***3000***.
2. Структура сервера должна соответствовать архитектуре N-layer.
3. Уровень моделей должен содержать описание моделей, соответствующих таблицам в ранее созданной БД, и связей между ними.
4. Уровень сервисов должен содержать бизнес-логику приложения и валидацию данных.
5. Сервис для *черепашек-ниндзя* должен предоставлять следующие операции:
6. Чтение всех черепашек-ниндзя.
7. Чтение черепашки-ниндзя по id, проверка является ли id числом.
8. Чтение всех черепашек-ниндзя, у которых любимая пицца *str* (задается клиентом).
9. Создание новой черепашки-ниндзя, проверять наличие всех полей (поля weaponId, favoritePizzaId, secondFavoritePizzaId, image могут быть пустыми).
10. Обновление черепашки-ниндзя по id с валидацией (описано выше).
11. Удаление черепашки-ниндзя по id.
12. Привязка любимой пиццы (первой и второй).
13. Отвязка любимой пиццы (первой и второй).
14. Привязка оружия.
15. Отвязка оружия.
16. Сервис для *оружия* должен предоставлять следующие операции:
17. Чтение всего оружия.
18. Чтение оружия по id, проверка является ли id числом.
19. Чтение всего оружия, у которого dps больше или меньше *n* (задается клиентом).
20. Создание нового оружия, проверять, чтобы dps был не более 500, а также наличие всех полей.
21. Обновление оружия по id с валидацией (описано выше).
22. Удаление оружия по id.
23. Сервис для *пицц* должен предоставлять следующие операции:
24. Чтение всех пицц.
25. Чтение пиццы по id, проверка является ли id числом.
26. Чтение всех пицц, у которых калорийность больше или меньше *m* (задается клиентом).
27. Создание новой пиццы, проверять, чтобы калорийность была не более 2000, а также наличие всех полей.
28. Обновление пиццы по id с валидацией (описано выше).
29. Удаление пиццы по id.
30. Уровень контроллеров должен принимать запросы клиентов и возвращать ответы.
31. Уровень роутов должен содержать следующие endpoint’ы:

|  |  |
| --- | --- |
| Метод | Маршрут |
| GET | api/turtles |
| GET | api/turtles/id |
| GET | api/turtles?favoritePizza=*str* |
| POST | api/turtles |
| PUT | api/turtles/ |
| PUT | api/turtles/favoritePizzaBind |
| PUT | api/turtles/secondFavoritePizzaBind |
| PUT | api/turtles/weaponBind |
| DELETE | api/turtles/favoritePizzaUnbind |
| DELETE | api/turtles/secondFavoritePizzaUnbind |
| DELETE | api/turtles/weaponUnbind |
| DELETE | api/turtles |
| GET | api/weapons |
| GET | api/weapons/id |
| GET | api/weapons?dps=gt *n*  api/weapons?dps=lt *n* |
| POST | api/weapons |
| PUT | api/weapons |
| DELETE | api/weapons |
| GET | api/pizzas |
| GET | api/pizzas/id |
| GET | api/pizzas?calories=gt *m*  api/pizzas?calories=lt *m* |
| POST | api/pizzas |
| PUT | api/pizzas |
| DELETE | api/pizzas |

|  |  |
| --- | --- |
| Метод  запроса | Назначение |
| GET | Возвращает 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. Для соединения с БД сервердолжен применять *пул соединений*.
2. Для проверки работоспособности приложения, используйте **POSTMAN**.

**Задание 04**

1. Разработайте HTML-страницу, на которой будет выводиться информация обо всех черепашках-ниндзя.
2. Список черепашек должен выводиться по 3 записи на странице. Внизу должны отображаться кнопки с номером страницы.

**Задание 05**

1. Фотографии черепашек-ниндзя хранить на жестком диске в директории images. В БД в поле image хранить полный путь к изображению.
2. В ответ на GET-запрос к /upload высылать HTML-страницу c формой, позволяющей отправить POST-запрос /upload. На форме должны находиться поле для ввода идентификатора черепашки-ниндзя, для которой добавляется изображение, элемент для выбора изображения для загрузки, а также кнопка для отправки формы на сервер.
3. Сервер должен возвращать ошибку, если черепашки-ниндзя с указанным идентификатором не существует, иначе же он должен сохранять файл в директорию images с названием «turtle\_*id*.jpg», где id – указанный идентификатор черепашки-ниндзя, а также добавлять соответствующую запись в БД**.**
4. Фотографии черепашек-ниндзя раздавать с сервера по адресу /images с использованием express.static.
5. В случае если запрошенной фотографии нет, возвращать ошибку 404.

**Задание 06**

1. Добавьте транзакцию, в которой происходит изменение всех пиццы с количеством калорий больше 1500 путем добавления к описанию «SUPER FAT!».
2. Продемонстрируйте работу транзакции.

**Задание 07.** Ответьте на следующие вопросы

1. Поясните понятие «ORM»
2. Поясните понятие «модель».
3. Поясните понятие «пул соединений» и его назначение.
4. Что такое Sequelize и Prisma?
5. Поясните, что такое «scope» в Sequelize?
6. Поясните, что такое «хуки» в Sequelize? Какие они бывают?
7. Поясните назначение функции Sequelize *Модель*.**hasMany**.
8. Поясните понятие «raw query» в Sequelize.
9. Назовите отличия Prisma от других фреймворков ORM?
10. Можно ли использовать Prisma с базами данных SQL и NoSQL?
11. Какие языки программирования совместимы с Prisma?
12. Расскажите, где и как можно использовать ***every***, ***none*** и ***some*** вPrisma?
13. Для чего нужен метод ***upsert*** вPrisma?
14. Как включить регистронезависимый поиск вPrisma?
15. Объясните назначение методов ***$queryRaw***, ***$executeRaw*** вPrisma.

27. \*\*ORM (Object-Relational Mapping)\*\* - это техника программирования, которая позволяет преобразовывать данные между несовместимыми типами систем, используя объектно-ориентированные языки программирования. В контексте баз данных ORM помогает преобразовывать данные между таблицами базы данных и объектами в коде, упрощая работу с базой данных.

28. \*\*Модель\*\* в контексте ORM - это абстракция, которая представляет таблицу в базе данных в виде класса в коде. Объекты этого класса соответствуют записям в таблице, а атрибуты класса - столбцам таблицы.

29. \*\*Пул соединений\*\* - это кэш соединений с базой данных, который поддерживает коллекцию активных соединений для их повторного использования. Назначение пула соединений - снижение накладных расходов, связанных с установлением соединений с базой данных, путем повторного использования существующих соединений, что улучшает производительность приложения.

30. \*\*Sequelize и Prisma\*\* - это два популярных ORM фреймворка для Node.js. Sequelize поддерживает обширный набор функциональностей для работы с реляционными базами данных, такими как MySQL, PostgreSQL и SQLite. Prisma - это современный ORM, который предоставляет мощный набор инструментов для работы с базами данных, включая миграции, генерацию клиентского кода и упрощенный запрос данных.

31. \*\*Scope в Sequelize\*\* - это предопределенный набор критериев запросов, которые можно применять к модели для упрощения запросов к базе данных. Scope можно использовать для создания фильтров или для задания постоянных параметров запроса.

32. \*\*Хуки в Sequelize\*\* - это функции обратного вызова, которые вызываются в определенные моменты жизненного цикла экземпляра модели (например, перед или после сохранения в базу данных). Они позволяют выполнять кастомную логику в эти моменты. Хуки бывают разных типов: `beforeValidate`, `afterValidate`, `beforeSave`, `afterSave` и т.д.

33. Функция \*\*Sequelize Модель.hasMany\*\* устанавливает отношение один ко многим между двумя моделями, указывая, что одна запись в родительской модели может быть связана с множеством записей в дочерней модели.

34. \*\*Raw query в Sequelize\*\* - это выполнение сырого SQL-запроса к базе данных напрямую, минуя абстракции ORM. Это полезно для выполнения сложных запросов, которые трудно или неэффективно выразить средствами ORM.

35. \*\*Отличия Prisma от других ORM\*\* включают более современный API, упрощенный доступ к данным с использованием Prisma Client, автоматическую генерацию схемы базы данных и клиентского кода, а также поддержку миграций через Prisma Migrate.

36. \*\*Prisma\*\* может использоваться с реляционными базами данных SQL, такими как PostgreSQL, MySQL, SQLite, и SQL Server. Поддержка баз данных NoSQL в Prisma ограничена и зависит от конкретных реализаций и требований проекта.

37. Prisma ориентирована на использование с \*\*JavaScript и TypeScript\*\*, предоставляя мощные инструменты для работы с базами данных в э

тих языках программирования.

38. Методы \*\*every, none, и some в Prisma\*\* позволяют выполнять фильтрацию данных на основе условий, связанных с отношениями между таблицами. Они используются для формирования сложных запросов, например, выбора всех пользователей, у которых есть хотя бы один определенный атрибут или ни одного.

39. Метод \*\*upsert в Prisma\*\* позволяет выполнить операцию вставки новой записи в базу данных, если она не существует, или обновить существующую запись, если она уже существует. Это удобный способ обновления данных без необходимости предварительной проверки их наличия.

40. Для \*\*регистронезависимого поиска в Prisma\*\* можно использовать фильтры, которые игнорируют регистр символов, такие как `contains`, `startsWith`, и `endsWith`, указав параметр `mode` как `insensitive`.

41. Методы \*\*`$queryRaw` и `$executeRaw` в Prisma\*\* позволяют выполнять сырые SQL-запросы напрямую к базе данных. `$queryRaw` используется для запросов, которые возвращают данные, в то время как `$executeRaw` применяется для выполнения запросов, не возвращающих данные (например, `UPDATE`, `DELETE`).