5. Создание пользовательских представлений и дальнейшего связыванием их с интерфейсом

В результате выполнения лабораторной работы на разрабатываемом веб-сайте должны заработать все кнопки и вместо зашитых в код данных будут отображены реальные данные из базы данных, полученные путем отправки сетевых запроса на сервер.

Для выполнения данной лабораторной работы вам необходимо реализовать в ранее созданном приложении, контроллеры и сервисы, которые были спроектированы в рамках предыдущей лабораторной работы.

Большинство современных веб-приложений предоставляют API, которые клиенты могут использовать для взаимодействия с приложением. Качественно спроектированный API должен поддерживать следующее:

- **Независимость от платформы**. Любой клиент должен иметь возможность вызывать API, независимо от того, как API реализован внутренне. Для этого требуется использование стандартных протоколов, а также наличие механизма, при котором клиент и веб-службы могут согласовать формат данных для обмена.

- **Развитие службы**. Веб-API должен иметь возможность развиваться и расширять набор функций независимо от клиентских приложений. По мере развития API имеющиеся клиентские приложения должны продолжать работать без изменений. Все функции должны быть доступными, чтобы клиентские приложения могли полноценно их использовать.

В рамках данной лабораторной работы от вас требуется реализовать всю серверную логику, для вашего проекта. т.е. добавить необходимую логику для создания/обновления/получения всех сущностей представленных на диаграмме сущностей.

Для корректной обработки всех входщих запросов рекомендуется использовать встроенные механизмы NestJS, например Guards. У guards есть единственная ответственность. Они определяют, будет ли данный запрос обработан обработчиком маршрута или нет, в зависимости от определенных условий (таких как разрешения, роли, ACL и т.д.), существующих во время выполнения. Рекомендуется проверять правильность любых данных, отправляемых в веб-приложение.

Реализовывать проверку входных данных рекомендуется с использованием привязки ValidationPipe на уровне приложения, чтобы обеспечить защиту всех конечных точек от получения неправильных данных. Если валидация реализована верно, то при отправке невалидного объекта ожидается ответ от сервера **HTTP 400 Bad Request.**

Следующим шагом необходимо убедиться на уровне контроллера в том, что объект переданный клиентов в post body является валидным, прежде чем пытаться запустить наш метод сервиса, иными словами необходимо добавить валидацию на основе схемы, спроектированной в лабораторной работе №3.

Вы можете это сделать это внутри метода обработчика маршрута, но такой подход не идеален, поскольку он нарушит правило одной ответственности (single responsibility rule - SRP). Но, рекомендуется реализовать для решения этой задачи свой **Validator**, единственной ответственностью которого будет либо пропустить запрос дальше в контроллер, либо выбросить ваш тип исключения.

Nest поставляется со встроенными уровнями исключений, которые отвечают за обработку всех необработанных исключений в приложении. Когда исключение не обрабатывается кодом вашего приложения, оно перехватывается этим уровнем, который затем автоматически отправляет соответствующий удобный ответ (например в HTTP 400 Bad Request). Реализуйте данную проверку с помощью Exception Filter’ов.

Важно! Убедитесь что добавленные аннотации корректно отображаются в сгенерированной Swagger документации. (см. пример на картинке ниже). Также убедитесь, что ваш фронтенд готов к тому, что сервер может вернуть как успешный ответ, так и ошибку.

POST `/store/order` Place an order for a pet

Parameters

Name	Description

body * required
object
(body)
	order placed for purchasing the pet
	Example Value | Model
	{
	"id": 0,
	"petId": 0,
	"quantity": 0,
	"shipDate": "2022-04-09T16:49:24.071Z",
	"status": "placed",
	"complete": true
	}
	Parameter content type
	application/json

Responses

Code	Description

200	successful operation
	Example Value | Model
	{
	"id": 0,
	"petId": 0,
	"quantity": 0,
	"shipDate": "2022-04-09T16:49:24.119Z",
	"status": "placed",
	"complete": true
	}

400	Invalid Order

Отправка большого объема данных через HTTP не очень хорошая идея. Безусловно, возникнут проблемы с производительностью, поскольку сериализация больших объектов JSON станет дорогостоящей. Best practice является разбиение результатов на части, а не отправка всех записей сразу. Предоставьте возможность разбивать результаты на странице с помощью предыдущих или следующих ссылок.

Если вы используете **пагинацию** в вашем приложении, одним из хороших способов указать ссылку на пагинацию является использование опции Link HTTP заголовка.