Разработка проекта началась с построения архитектуры. Приложения на NESTJS имеют модульную архитектуру. На слайде представлен один из модулей приложения, а именно модуль “post”. Помимо стандартных файлов модуля, были добавлены также файлы для dto, entities, interfaces, response и types. Все модули в основном имеют примерно одинаковые состав, за исключением модулей логирования и авторизации.  
Следующая часть архитектуры это директория “common”. Она содержит файлы для всего проекта.

Директория Entities содержит сущности приложения, которые не имею собственных модулей, а в helpers содержатся различные вспомогательные функции.

Для аутентификации в проекте использовалась библиотека passport. Passport имеет богатую экосистему стратегий, которая реализует различные механизмы аутентификации. Стратегии представляют собой классы, которые определяют, как будет происходить процесс аутентификации и какие данные будут проверяться для авторизации пользователя. На слайде представлена реализации JWT стратегии. Она используется посредством декоратора UseGuards для запросов, которые требуют аутентификации. Также на слайде представлен запрос на регистрацию пользователя…

Который вызывает метод создания в UserService и в конечном счёте вызывается метод репозитория, в котором запись добавляется в базу данных. При работе над реализацией аутентификации я изучил много информации и применил лучшие практики безопасности, а именно использование JWT и вместо хранения паролей в открытом виде хеширование с помощью соли. Также есть реализация рефреш токена.

В проекте реализация логирования с помощью библиотеки Winston, которая предоставляет мощный инструмент для отслеживания и анализа работы приложения. На слайде представлена реализация класса логирования. Содержимое логов и формат записи должны быть различным для разных окружений, что и реализовано в данном классе. Для логирования содержимого запросов был создан middleware и подключён в классе AppModule для всех запросов.

Для логирования ошибок был создан класс, наследуемый от ExeptionFilter.

Для логирования ответов был разработан класс-перехватчик, который непосредственно перехватывает ответ перед отправлением клиенту и логирует данные.  
Сложность реализации этого задания заключалась в том, что я не имел опыта работы с фильтрами и перехватчиками. В результате изучив их и библиотеку winston, мне удалось реализовать логирование приложения.

Создание crud операций представлено на примере модуля post. В контролер приходит запрос за какую-либо операцию и если это операция не чтения данных, то для выполнения необходимо, чтобы пользователь был авторизован. Из контроллера вызывается метод сервиса, где реализована бизнес-логика и взаимодействие с репозиторием.

В самом классе репозитория написана логика для взаимодействия с данными. Сложностью этого задания для меня стало определение целесообразности запросов. Я думаю некоторые запросы требуют уточнения, в плане возвращаемых данных. Также я считаю, что для конечного продукта можно добавить несколько запросов, но на данной стадии разработки для меня сложно определить, какие запросы будут полезны, а какие могут стать лишними.