GALLERY

Экосистема:

Требуется инициализировать базовое приложение (используя любой готовый Boilerplate вроде CRA для Frontend части и NestJS CLI для старта Backend части или свои наработки). На личном GitHub-аккаунте создать публичный репозиторий с названием этого задания (Gallery) и возможность создавать pull request любыми лицами.

Суть приложения: галерея изображений/видео пользователей с возможностью раздачи permissions.

Backend:

1. Авторизация – sign-up, sign-in, sign-out, refresh-token, verify-email (использовать nodemailer для отправки сообщений).
2. Permissions – есть дефолтное состояние:
   - пользователь только зарегистрировался в систему (это значит, что его видео доступны всем);
   - у пользователя есть возможность запретить какому-то пользователю просматривать его галерею (это значит, что никакое изображение/видео не будет доступно для этого пользователя);
   - у пользователя есть возможность закрыть определенное изображение/видео для конкретного пользователя (это значит, что при просмотре галереи пользователя, запрещенное видео не должно быть доступно для просмотра);
   - у пользователя есть возможность запретить просмотр изображений/видео для всех.
3. Admin, internalUser. Если я Admin, то у меня есть привилегии для удаления/редактирования видео/изображений или изменении permissions для любого пользователя.

Frontend:

1. Авторизация/регистрация; Sign in и Register – это 2 разные страницы, соответственно разные роуты в приложении.

2. Просмотр ленты последних опубликованных мной видео/изображений. На данной странице реализовать https://mui.com/material-ui/react-drawer/ меню подобного формата для перехода на свою страницу, на страницу настроек(в которых можно будет управлять permissions).

3. Permissions. Два списка:

– в первом списке пользователи и какое конкретное видео запретили;

– во втором списке пользователи, которым запрещен полный просмотр всех видео.

4. Поиск пользователей по имени. Просмотр ленты конкретного пользователя.

В качестве ориентировочного дизайна можно использовать пример мобильного приложения, приложенный внизу описания. Разумеется, это приложение надо адаптировать под веб-приложение (в произвольной форме, но я бы оставлял такую же структуру и увеличивал max-width страницы до 762 пикселей).

Все состояние приложения, сохранение и обновление данных, авторизацию надо реализовать через Backend API. Для хранения изображений использовать File Storage вашей локальной машины, в базе данных хранить путь до изображений/видео, находящихся на вашей локальной машине.

ВАЖНО: все secret-значения не должны храниться в репозитории, их надо выносить в .ENV-файл и добавлять в .GITIGNORE репозитория.

Технические требования:

1. Требуется использование React-Redux и Redux-Saga для менеджмента данных.

2. Требуется использование TypeScript.

3. Для Backend использовать NestJS. Если вы выбираете MongoDB, то использовать Mongoose как ORM, если Postgres, то TypeORM, Функционал миграций так же должен быть реализован (не использовать synchronize: true).

4. Использовать NestJS подход для реализации Backend функционала. Controller -> Service -> Repository.

5. Для роутинга в приложении в React можно использовать сторонние библиотеки. Важно: если пользователь не авторизовался, его не должно пустить на страницу создания или просмотра задач.

6. Перед разработкой надо настроить ESLint для приложения. В качестве дополнения можно настроить Prettier и связать его с ESLint. Будет дополнительным плюсом реализовать pre-commit hook, который не позволит сделать push в репозиторий, если в приложении присутствуют ошибки ESLint.

7. Грамотная работа с бд. По возможности с клиента должно идти как можно меньше запросов. Банальный пример не самого оптимального использования: делать отдельный запрос на задачи для каждого дня (т.е. для 30 дней месяца улетают 30 запросов).

8. Написание документации к проекту; в README-файле проекта перед сдачей задания должна быть написана краткая документация на английском или русском, состоящая из 4 пунктов:

   – “Task” (ссылка на этот документ);

   – “How to run the app” (инструкции по установке модулей и запуску приложения);

   – “Database snapshot” (надо показать, как в базе данных организована работа с сущностями);

   – “Application stack” (описание технологического стека, использованного в приложении: какие дополнительные библиотеки и утилиты, были использованы).

9. Обработка ошибок с сервера: если пользователь пытается авторизоваться с неправильными данными – Backend API вернет ошибку, и это надо вывести пользователю на экран в виде каких-либо Toast (для этого для скорости реализации можно использовать какую-нибудь стороннюю библиотеку).

Дополнительные баллы можно получить за:

1. Действительно удобный пользовательский интерфейс.

2. Реализованный Theme-менеджмент. Возможность с легкостью поменять цветовую схему приложения **из кода** (не надо реализовывать переключатели для пользователя).

3. Оптимизированные запросы в бд.

4. Правильное отделение бизнес логики от слоя представления данных (controllers) в Backend части приложения.

5. Корректную обработку ошибок на серверной стороне, а так же логирование ошибок в файл.