Тестовое задание для Fullstack-разработчика

Цель задания – выяснить навыки и возможности создания соискателем Web-приложений состоящих из сервера NodeJS, клиентской части на React и СУБД Postgres.

В результате выполнения должно быть получено работоспособное Web-приложение, написанное на Javascript (без использования TypeScript). Выполняющим задание нужно иметь в виду, что код будет внимательно рассмотрен на качество написания, используемые конструкции, использованные компоненты и средства.

Если в полученном Web-приложении будут ошибки, небезопасный код (без обработки исключений), лишние (избыточные) конструкции или задание будет сделано не верно (не полностью), то такой результат не будет принят.

Задание:

1. Создать базу данных в СУБД Postgres с двумя таблицами. Таблицы связать по полю foreign key. Каждая таблица должна иметь поле-ключ и другие поля типа NUMERIC, VARCHAR, DATE, INTEGER. Имена БД и таблиц выбрать произвольно и заполнить произвольными данными. Создание БД и ее заполнение начальными данными написать на языке SQL в виде скрипта и поместить его в отдельный файл init-db.sql в корневую папку проекта.
2. Создать серверную часть с использованием NodeJS. Подключиться к созданной в п.1 БД с помощью средств pg-promise (https://github.com/vitaly-t/pg-promise) или sequalize (https://sequelize.org/).

   2.1. В случае использования sequelize написать часть функций (методов) работы с данными на языке SQL (использовать метод seq.query()).

   При реализации частичной подгрузки данных (пагинации, infinite loading) использовать операторы языка SQL limit и offset.

   2.2. Роутеры написать по архитектурному стилю REST API. Реализовать базовые операции: создание, получение, изменение и удаление.

   2.3. Серверная часть должна включать папки: **модели (/models), контроллеры (/controllers), роутеры (/routes)**.

   Все сущности должны быть разбиты по данной структуре. Например, для сущности Книга(book) – это будет 3 файла:

   - Роутер, где описаны маршруты по архитектурному стилю REST API (примеры: GET /api/books, POST /api/books/:id);
   - Контроллер, который вызван из роутера. В нем могут быть различные проверки данных и дальнейший вызов методов модели;
   - Модель, которая реализует методы работы с БД.

3. Создать клиентскую часть с использованием React.

   Внешний вид оформить в виде Dashboard с боковым меню из нескольких произвольных пунктов. В первом пункте поместить 2 таблицы Ag-Grid (https://www.ag-grid.com/), в которые вывести данные из таблиц созданных в п.1 с помощью сервера из п.2.

   Над **связанной** таблицей поместить работающие кнопки: Добавить запись, Изменить, Удалить. При добавлении/изменении записи учитывать связь с главной таблицей в поле с foreign key.

   3.1. В главной таблице применить загрузку с помощью **infinite loading** (использовать AgGrid Infinite Row Model). Обязательно реализовать частичную подгрузку записей с сервера (использовать операторы языка SQL limit и offset).

   3.2. При оформлении Dashboard использовать шаблон одного из современных примеров с сайта Creative Tim раздела React FREE (https://www.creative-tim.com/templates/react), либо любой другой, аналогичный по виду и структуре (красиво сверстанный, со стилем, взятым из него). Разрешается взять готовый и поменять под тестовое задание.

4. Выложить на GitHub:
   - исходные тексты Web-приложения (без папки node_modules). Клиентская и серверная части должны находиться в одном репозитории в отдельных папках в корне проекта (/client и /server);
   - sql-скрипт для создания БД с данными (в файле init-db.sql);
   - инструкцию по установке и запуску (в файле Readme.md расположенном в корне проекта).

Срок сдачи: 1-2 недели.

Желаем успехов и ждем вашу работу!