**Тестовое задание для вакансии Loymax AI Data Scientist (LLM/NLP)**

**1. Цель задания:**

Оценить практические навыки кандидата в проектировании, разработке и развертывании end-to-end ML-решения (Retrieval-Augmented Generation - RAG), с акцентом на исследование данных, качество кода, модульность, контейнеризацию, тестирование и подготовку к промышленной эксплуатации.

**2. Задача:**

Разработать прототип сервиса вопрос-ответ (Q&A) на основе RAG архитектуры. Система должна:

\* Принимать текстовый вопрос пользователя.

\* Находить наиболее релевантные фрагменты информации из предоставленного набора документов (knowledge base).

\* Использовать LLM для генерации полного, информативного ответа на основе найденных фрагментов.

\* Быть реализована как набор взаимодействующих микросервисов/компонентов, готовых к контейнеризации и развертыванию.

\* Иметь потенциал для горизонтального масштабирования.

**3. Основные Требования к Реализации:**

**3.1. Исследование и Подготовка Данных:**

\* **Данные:** Использовать предоставленный датасет (https://raw.githubusercontent.com/vladislavneon/RuBQ/refs/heads/master/RuBQ\_2.0/RuBQ\_2.0\_paragraphs.json)

\* **Анализ**: Провести базовый анализ данных. Результаты кратко описать в README.

\* **Preprocessing:** Реализовать скрипт/модуль предобработки текста (очистка, нормализация и т.д.). Код должен быть модульным и конфигурируемым.

\* **Векторизация:** Выбрать и применить модель для создания эмбеддингов текстов (например, любая подходящая модель с Hugging Face). Сохранить эмбеддинги в векторной базе данных.

\* **Векторная БД**: Использовать легковесную векторную БД, подходящую для прототипа и поддерживающую контейнеризацию (например, ChromaDB, FAISS (в режиме persistence)). Код инициализации/наполнения БД должен быть частью решения.

**3.2. Проверка Качества Данных (Data Quality Checks):**

\* Реализовать базовые проверки **перед** созданием эмбеддингов и сохранением в векторную БД:

\* Проверка на пустые документы.

\* Проверка на дубликаты (по `id` и/или хешу `text`).

\* Проверка на минимальную длину текста (для отсечения "мусора").

\* (Опционально, но приветствуется): Проверка кодировки, наличие "битых" символов.

\* Результаты проверок должны логироваться. Документы, не прошедшие проверки, должны либо отфильтровываться, либо явно обрабатываться (решение и обоснование описать в README).

**3.3. Основная Кодовая База (Микросервисы/Компоненты):**

Система должна быть разделена как минимум на 2 основных сервиса (можно реализовать в виде отдельных скриптов/приложений, взаимодействующих через API или очереди):

\* **Сервис Индексирования (Indexing Service):**

\* Принимает путь к сырым данным (или каталог).

\* Запускает пайплайн: загрузка -> проверка качества -> препроцессинг -> векторизация -> сохранение в векторную БД.

\* Должен быть пригоден для повторного запуска (инкрементальное обновление индекса - большой плюс).

\* **Сервис Запросов (Query/Inference Service):**

\* Предоставляет API endpoint (например, FastAPI или Flask) для приема вопросов (`/query`).

\* Пайплайн обработки запроса: препроцессинг вопроса -> поиск релевантных чанков в векторной БД -> формирование промпта для LLM -> запрос к LLM -> возврат сгенерированного ответа.

\* Использовать LLM с Hugging Face Hub. Использование `Inference API` или локального запуска (e.g., через `transformers` pipeline) - на выбор кандидата, с обоснованием в README.

\* **Конфигурация:** Использовать конфигурационные файлы (`.env`, `config.yaml`, `config.py`) для управления путями, параметрами моделей, настройками векторной БД и т.д.

**3.4. Упаковка в Контейнеры:**

\* Для каждого основного сервиса (Индексирование, Запрос) создать отдельный `Dockerfile`.

\* Настроить `docker-compose.yml` для оркестрации сервисов: Сервис Запросов, Векторная БД (если используется как отдельный контейнер, например Qdrant), и опционально Сервис Индексирования. Убедиться в корректности сетевых взаимодействий.

\* Контейнеры должны собираться и запускаться без ошибок командой `docker-compose up --build`.

**3.5. Тестирование:**

\* Реализовать **unit-тесты** (с помощью `pytest` ) для критических модулей, таких как:

\* Функции препроцессинга текста.

\* Логика формирования промпта на основе вопроса и релевантных чанков.

\* (По возможности) Моки для тестирования взаимодействия с векторной БД и LLM.

\* Реализовать **интеграционный тест** (или скрипт), проверяющий работу всего пайплайна Q&A после запуска `docker-compose up` (например, отправка тестового вопроса через API и проверка наличия ответа).

**3.6. Развертывание и Масштабируемость:**

\* Решение должно быть **готово к развертыванию** на сервере через `docker-compose up`.

\* Продемонстрировать **потенциал масштабируемости** в архитектуре:

\* Четкое разделение на сервисы (Indexing, Query, Vector DB).

\* Использование конфигурации для легкой замены компонентов (моделей, БД).

\* В `README.md` описать, как можно масштабировать систему (например, запуск нескольких реплик Сервиса Запросов за балансировщиком нагрузки, использование более производительной векторной БД в кластере).

**3.7. Документация и Код:**

\* **README.md (Техническая Документация):** В корне репозитория. Должен содержать:

1. Описание проекта и его цели.

2. Архитектурная схема (блок-схема компонентов и их взаимодействия).

3. Инструкция по установке зависимостей (`requirements.txt` или аналогичный обязателен).

4. Инструкция по запуску:

\* Локально (установка, запуск индексации, запуск API).

\* Через Docker (`docker-compose up`).

5. Описание использованных технологий, моделей (LLM, эмбеддинги), векторной БД с обоснованием выбора.

6. Описание API endpoint'а Сервиса Запросов (формат запроса/ответа).

7. Результаты базового анализа данных и подход к проверке их качества.

8. Описание реализованных тестов и как их запустить.

9. Описание возможных путей улучшения и масштабирования системы.

\* **Качество Кода:**

\* Строгое соответствие **PEP 8.**

\* Чистый, модульный, документированный код (docstrings для ключевых функций/классов).

\* Осмысленные названия переменных, функций, классов.

\* Логирование ключевых событий.

\* Обработка основных ошибок.

\* **Структура Репозитория:** Четкая и логичная (например: `src/`, `api/`, `indexing/`, `vector\_db/`, `tests/`, `data/`, `dockerfiles/`, `configs/`, `README.md`, `requirements.txt`, `docker-compose.yml`).

**4. Требования к Представлению Результатов:**

\* Кандидат должен предоставить ссылку на **приватный Git-репозиторий** (GitHub, GitLab, Bitbucket) с полным кодом проекта.

\* Кандидат должен быть готов на собеседовании:

\* Кратко презентовать решение (архитектура, ключевые решения, использованные технологии).

\* Запустить систему локально или из контейнеров и продемонстрировать ее работу (ответ на примерные вопросы).

\* Ответить на вопросы по коду, архитектурным решениям, тестам, данным, ограничениям системы и возможным улучшениям.

\* Обосновать выбор конкретных моделей, библиотек и подходов.

\* Показать подготовленные материалы (архитектурная схема, README).

**5. Сроки и Дополнительно:**

\* Ожидаемое время выполнения: **5-7 дней**.

\* Использование облачных провайдеров (AWS, GCP, Azure) или продвинутых оркестраторов (K8s) **не требуется** для этого задания, фокус на локальную работу и контейнеризацию через Docker Compose.

\* Приветствуется использование современных, но стабильных библиотек (LangChain/LlamaIndex - **допустимо, но не обязательно**, чистая реализация на `transformers`, `sentence-transformers`, `fastapi`/`flask` также отличный вариант).

\* Критерии оценки: Качество кода (PEP8, структура), работоспособность end-to-end пайплайна, корректность контейнеризации, наличие и качество тестов, полнота и ясность документации (README), обоснованность принятых решений, потенциал масштабируемости, качество презентации и защиты.

Удачи в выполнении задания! Мы ожидаем увидеть ваш подход к решению реальной инженерной задачи.