Необходимо написать приложение

Стек и тулы:

1. База данных Postgres
2. Использование Redis/Elastic и т.д. для определенных целей – на ваше усмотрение.
3. Java 11 версии
4. Имплементация на Spring boot.
5. REST API.
6. Сборка: Maven

Структура таблиц БД (если резко необходимо, то можно добавить поля и таблицы)

Key type	Column	Type	Unique	Comment
Table: **USER**				
PK	ID	BIGINT	True	
	NAME	VARCHAR(500)		
	DATE_OF_BIRTH			Format: 01.05.1993
	PASSWORD	VARCHAR(500)		Min length: 8, max 500
Table: **ACCOUNT**				
PK	ID	BIGINT	True	
FK	USER_ID	BIGINT	True	Link to USER.ID
	BALANCE	DECIMAL		рубли + копейки: в коде – BigDecimal
Table: **EMAIL_DATA**				
PK	ID	BIGINT	True	
FK	USER_ID	BIGINT		Link to USER.ID
	EMAIL	VARCHAR(200)	True	
Table: **PHONE_DATA**				
PK	ID	BIGINT	True	
FK	USER_ID	BIGINT		Link to USER.ID
	PHONE	VARCHAR(13)	True	format: 79207865432

Общие требование к системе:

1. 3 слоя – API, service, DAO
2. Будем считать, что в системе только обычные пользователи (не админы и т.д).
3. Будем считать, что пользователи создаются миграциями (не будем усложнять). Просто считаем, что для обычных пользователей нет операции создания. Для тестов создать напрямую в DAO.
4. У пользователя может быть более одного PHONE_DATA (должен быть как минимум 1).
5. У пользователя может быть более одного EMAIL_DATA (должен быть как минимум 1).
6. У пользователя должен быть быть строго один ACCOUNT.

7. Начальный BALANCE в ACCOUNT указывается при создании пользователя.
8. BALANCE в ACCOUNT не может уходить в минус ни при каких операциях.
9. Валидация входных API данных.

Обязательные фичи:

1. CREATE (только для определенных внутри пользователя данных), UPDATE операции для пользователя. Пользователь может менять только собственные данные:
    a) может удалить/сменить/добавить email если он не занят другим пользователям
    b) может удалить/сменить/добавить phone если он не занят другим пользователям
    c) остальное менять не может
2. Реализовать READ операцию для пользователей. Сделать «поиск пользователей» (искать может любой любого) с фильтрацией по полям ниже и пагинацией (size, page/offset):
    a) Если передана «dateOfBirth», то фильтр записей, где «date_of_birth» больше чем переданный в запросе.
    b) Если передан «phone», то фильтр по 100% сходству.
    c) Если передан «name», то фильтр по like форматом ‘{text-from-request-param}%’
    d) Если передан «email», то фильтр по 100% сходству.
2. Добавить JWT token (необходимый Claim только USER_ID), механизм получения токена на ваше усмотрение. Имплементировать максимально просто, не стоит усложнять. Аутентификация может быть по email+password, либо по phone+password.
3. Раз в 30 секунд BALANCE каждого клиента увеличиваются на 10% но не более 207% от начального депозита.
   Например:
   Было: 100, стало: 110.
   Было: 110, стало:121.
   …
4. Сделать функционал трансфер денег от одного пользователя к другому.
   Вход: USER_ID (transfer from) – берем у авторизованного из Claim токена, USER_ID (transfer to) из запроса, VALUE (сумма перевода) из запроса.
   То есть у того, кто переводит мы списываем эту сумму, у того, кому переводим – добавляем эту сумму.
   Считать эту операцию «банковской» (высоко-значимой), сделать ее со всеми нужными валидациями (надо подумать какими) и потоко-защищенной.

Не обязательные фичи (но которые очень хотелось бы видеть, хотя бы в минимальном исполнении):

1. Добавить swagger (минимальную конфигурацию).
2. Добавить не бездумное (значимое) логгирование.
3. Добавить корректное кэширование (например на API и на DAO слой). Имплементация на ваше усмотрение.

Тестирование:

Покрытие unit тестами. Не надо покрывать тестами весь код. Нужно сделать тесты на покрытие функционала трансфера денег и какую-то API операцию покрыть через testcontainers с использованием MockMvc.

Результат:

Результат тестового задания необходимо предоставить в виде ссылки на публичный репозиторий на Гитхабе.

Реализация frontend не обязательна,

Желаем удачи в прохождении тестового задания!