**Срок выполнения:** 4 суток с момента получения задания. Если нужно больше времени по каким-либо причинам, просьба предупредить сразу.

**В README обязательно укажите:**

Инструкции по запуску проекта.

Описание использованных технологий (если выходили за рамки базового стека).

Логику принятых решений (если были нестандартные подходы).

**Необходимо написать приложение**

**Стек и тулы:**

1. База данных Postgres
2. Java 11 версии и выше
3. Имплементация на Spring boot.
4. REST API.
5. Сборка: Maven
6. (опционально) для оптимизации поиска можете использовать Elasticsearch, а для реализации слоя кэширования приложения – Redis или аналогичный инструмент
7. Использование иных инструментов для определенных целей – на ваше усмотрение

**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. Добавить корректное кэширование (например на API и на DAO слой). Имплементация на ваше усмотрение.
2. CREATE (только для определенных внутри пользователя данных), UPDATE операции для пользователя. Пользователь может менять только собственные данные:
   1. может удалить/сменить/добавить email если он не занят другим пользователям
   2. может удалить/сменить/добавить phone если он не занят другим пользователям
   3. остальное менять не может
3. Реализовать READ операцию для пользователей. Сделать «поиск пользователей» (искать может любой любого) с фильтрацией по полям ниже и пагинацией (size, page/offset):
   1. Если передана «dateOfBirth», то фильтр записей, где «date\_of\_birth» больше чем переданный в запросе.
   2. Если передан «phone», то фильтр по 100% сходству.
   3. Если передан «name», то фильтр по like форматом ‘{text-from-request-param}%’
   4. Если передан «email», то фильтр по 100% сходству.
4. 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.
5. 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.  
…*

1. 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.

**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!**