**Тестовое задание Junior Java разработчика**

**Цель**: Кандидату предлагается написать микросервис, состоящий из двух API, одного клиента для обращения к внешнему API, и использующего как минимум одну базу данных для хранения бизнес-сущностей и курсов валют.

Данной задачей подразумевается проверка написания эффективных алгоритмов, умение строить архитектуру приложения, умение работать с SQL (в рамках задачи необходимо применять JOIN с подзапросом и агрегирующей функцией), пользоваться WEB-фреймворками, при желании можно использовать средства пакета *java.util.concurrent* для распараллеливания расчетов транзакций в нескольких валютах, получению актуальных курсов валют. Полный список критериев оценки приведен ниже.

Java:

* BigDecimal, Date types, enums
* Collections, equals and hashCode, Stream API, Lambdas
* Обработка исключений
* Generics
* Code conventions
* **Multithreading\*, Reflection API\***

Фреймворки, библиотеки и Build Tools:

* Spring WEB, WebFlux: бины, компоненты, конфигурации, Jackson lib
* ORM (Spring Data, JOOQ): Entities, Transactions, Relations (ManyToOne, OneToMany), JPQL, native queries, data mappings
* HTTP clients: Feign, Reactive Feign, WebClient
* **DTO-Entity mappers: MapStruct, JMapper**
* Конфигурации, @ComponentScan, Spring Boot автоконфигурация с исключениями\*
* Lombok
* Maven/Gradle (Bill of Materials, dependency management, build plugins, модули)

БД:

* Язык SQL, нормальные формы, отношения таблиц, внешние ключи, подзапросы
* NoSQL и их уместное применение (Mongo, Cassandra, Clickhouse, внешние кэши) \*
* Инструменты миграции (Flyway, Liquibase)

Общие инженерные навыки:

* Понимание HTTP-протокола
* Архитектура приложения (разделение на компоненты (Controller-Service-Repository) и пакеты, выделение Utils-методов в отдельные классы)
* Разработка алгоритмов (полиномиальных, желательно эффективнее O(n2))
* Использование паттернов и избегание антипаттернов
* **Работа с документацией (внешний API), написание собственной (JavaDoc, Swagger, README.md)**
* Чистота кода (следование принципу DRY, исключение хардкода, работа с переменными окружения и конфигурациями для различных сред (dev, test, prod)
* Следование принципам SOLID, DDD
* **Логирование: info, debug, warning, error; logback.xml; logbook dependency**
* Работа с системой контроля версий
* Умение дать оценку по времени выполнения и уложиться в срок

QA:

* **Unit-тесты методов бинов и Utils-классов**
* **Интеграционные тесты, использование моков \***
* **Тестовая конфигурация и test-stage в CI pipeline \***

Infrastructure and Linux:

* **Docker, Docker Compose**
* **CI/CD pipelines, Bash \*, Развертывание кода на публичном сервере опционально \***

**Описание задания**:

Вы работаете Junior Java разработчиком в банке. Поступает следующая задача от руководителя:

Требуется разработать прототип микросервиса, без разграничений доступа к API, который будет интегрирован в существующую банковскую систему. Микросервис должен:

1. Получать информацию о каждой расходной операции в тенге (KZT), рублях (RUB) и других валютах в реальном времени и сохранять ее в своей собственной базе данных (БД);
2. Хранить месячный лимит по расходам в долларах США (USD) раздельно для двух категорий расходов: товаров и услуг. Если не установлен, принимать лимит равным 1000 USD;
3. Запрашивать данные биржевых курсов валютных пар KZT/USD, RUB/USD по дневному интервалу (1day/daily) и хранить их в собственной базе данных. При расчете курсов использовать данные закрытия (close). В случае, если таковые недоступны на текущий день (выходной или праздничный день), то использовать данные последнего закрытия (previous\_close);
4. Помечать транзакции, превысившие месячный лимит операций (технический флаг limit\_exceeded);
5. Дать возможность клиенту установить новый лимит. При установлении нового лимита микросервисом автоматически выставляется текущая дата, не позволяя выставить ее в прошедшем или будущем времени. Обновлять существующие лимиты запрещается;
6. По запросу клиента возвращать список транзакций, превысивших лимит, с указанием лимита, который был превышен (дата установления, сумма лимита, валюта (USD)).

Для выполнения п.3:

Рассчитывать сумму расходов в USD нужно по биржевому курсу на день расхода или по последнему курсу закрытия.

Данные биржевых торгов получать из внешнего источника данных ([twelvedata.com](https://twelvedata.com/docs#time-series), [alphavantage.co](https://www.alphavantage.co/), [openexchangerates.org](https://docs.openexchangerates.org/reference/convert) или из другого по своему усмотрению).

За каждый запрос внешних данных нужно платить, и, к тому же, на выполнение внешнего запроса тратится дополнительное время. В связи с этим, полученные обменные курсы валют нужно хранить в своей базе данных и преимущественно использовать их.

Для выполнения п.4:

Последний лимит не должен влиять на выставление флага limit\_exceeded транзакциям, совершенным ранее установления последнего лимита. Иными словами, если лимит, установленный 1.01.2022 в размере 1000 USD, превышен двумя транзакциями на суммы 500 и 600 USD, то второй транзакции должен быть выставлен флаг limit\_exceeded = true. Если пользователь установил новый лимит 11.01.2022, и выполнил третью транзакцию 12.01.2022 на сумму 100 USD, она должна иметь флаг limit\_exceeded = false.

Следующая таблица демонстрирует логику выставления флагов транзакциям, но не является решением, которое должно быть реализовано в рамках разработки модели БД:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Дата установления лимита** | **Лимит, USD** | **Остаток месячного лимита, USD** | **Дата совершения транзакции** | **Сумма транзакции** | **limit\_exceeded** |
| **1 случай** | | | | | |
| 1.01.2022 | 1000 | 1000 |  |  |  |
|  |  | 500 | 2.01.2022 | 500 | false |
|  |  | -100 | 3.01.2022 | 600 | true |
| 10.01.2022 | 2000 | 900 |  |  |  |
|  |  | 800 | 11.01.2022 | 100 | false |
|  |  | 100 | 12.01.2022 | 700 | false |
|  |  | 0 | 13.01.2022 | 100 | false |
|  |  | -100 | 13.01.2022 | 100 | true |
| **2 случай** | | | | | |
| 1.02.2022 | 1000 | 1000 |  |  |  |
|  |  | 500 | 2.01.2022 | 500 | false |
|  |  | 400 | 3.01.2022 | 100 | false |
| 10.02.2022 | 400 | -200 |  |  |  |
|  |  | -300 | 11.01.2022 | 100 | true |
|  |  | -400 | 12.01.2022 | 100 | true |

Для выполнения п.6:

По запросу пользователя, ориентируясь на таблицу выше, микросервис должен вернуть список из двух транзакций, выполненных 3 и 13 января в первом случае и две транзакции за 11 и 12 января во втором случае.

Подсказка:

При получении лимитов, в SQL запросе пользуйтесь JOIN с подзапросом, агрегирующими функциями и группировками.

Структура данных транзакции на входе в сервис:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **п/п** | **Наименование параметра** | **Имя параметра** | **Тип данных** | **Пример** |
| 1 | Банковский счет клиента | account\_from | Целочисленный, 10 знаков | 0000000123 |
| 2 | Банковский счет контрагента | account\_to | Целочисленный, 10 знаков | 9999999999 |
| 3 | Валюта счета | currency\_shortname | Строка | KZT |
| 4 | Сумма транзакции | sum | Число с плавающей точкой, округление до сотых | 10000,45 |
| 5 | Категория расхода | expense\_category | Строка | product / service |
| 6 | Дата и время | datetime | Отметка времени с временной зоной | 2022-01-30 00:00:00+06 |

Структура данных ответа из п.6 аналогична приведенной выше структуре, за одним исключением: к каждой транзакции должны быть добавлены следующие три параметра:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **п/п** | **Наименование параметра** | **Имя параметра** | **Тип данных** | **Пример** |
| 7 | Сумма установленного лимита | limit\_sum | Число с плавающей точкой, округление до сотых | 1000.00 |
| 8 | Дата и время установления лимита | limit\_datetime | Отметка времени с временной зоной | 2022-01-10 00:00:00+06 |
| 9 | Валюта лимита | limit\_currency\_shortname | Строка | USD |

**Требования**

1. **Сервис должен содержать два REST/SOAP/GraphQL (на выбор) API:**

А) для приема транзакций (условно интеграция с банковскими сервисами);

Б) клиентский, для внешних запросов от клиента: получение списка транзакций, превысивших лимит, установление нового лимита, получение всех лимитов.

1. Покрыть документацией как минимум API, в README.md описать шаги запуска сервиса. Для описания API можно воспользоваться Swagger или аналогами.
2. Выбрать БД для хранения сущностей: PostgreSQL, MySQL, или иную. Модель данных разработать самостоятельно.
3. Скрипты миграций БД привести в том же репозитории. Крайне желательно воспользоваться Liquibase/FlyWay и **подходом schema first.**
4. Покрыть код unit и/или интеграционными тестами, обязательно покрыть тестами логику выставления флагов limit\_exceeded.
5. Опубликовать в публичном репозитории в GitHub/GitLab.
6. Ознакомиться с задачей и заявить необходимое время на выполнение задания – таким образом проверяется навык оценки трудоемкости задачи.

**Необязательные требования**

Выполняются по желанию, но будут однозначно расценены положительно:

1\*. **Реализовать параллельное выполнение алгоритма для транзакций клиента в различных валютах.**

2\*. **Для хранения курсов валют предпочтительно использовать NoSQL DB Cassandra.**

3\*. Подготовить сервис к запуску в Docker; желателен скрипт Docker Compose.

4\*. Написать CI pipeline для выбранной системы контроля версий.

**5\*. Запустить сервис на публичном сервере и приложить ссылку на API.**