Разработать RESTful web-service, реализующей функционал для работы с банковскими сущностями клиента.

Основные сущности:

* **Счет**
* **Карта**
* **Кредит**
* **Депозит**

Наполнение данными происходит из exchange в RabbitMQ. Примеры возможных сообщений:

{

"header": {

"message\_type": "account\_info"

},

"payload": {

"name": "Название счёта",

"iban": "AABBCCCDDDDEEEEEEEEEEEEEEEE",

"iban\_readable": "AABB CCC DDDD EEEE EEEE EEEE EEEE",

"amount": 2100.00,

"currency\_code": "933",

"open\_date": "dd.MM.yyyy",

"main\_acc": true,

"customer\_id": "1a72a05f-4b8f-43c5-a889-1ebc6d9dc729",

"customer\_type" : "LEGAL",

"rate": 0.01

}

}

{

"header": {

"message\_type": "card-info"

},

"payload": {

"card\_number": "5200000000001096",

"card\_number\_readable": "5200 0000 0000 1096",

"iban": "AABBCCCDDDDEEEEEEEEEEEEEEEE",

"customer\_id": "1a72a05f-4b8f-43c5-a889-1ebc6d9dc729",

"customer\_type" : "LEGAL/PHYSIC",

"cardholder": "CARDHOLDER NAME",

"card\_status": "ACTIVE/INACTIVE/BLOCKED/NEW"

}

}

{

"header": {

"message\_type": "credit-details"

},

"payload": {

"customer\_id": "1a72a05f-4b8f-43c5-a889-1ebc6d9dc729",

"contractNumber": "11-0216444-2-0",

"contractStartDate": "30.03.2022",

"totalDebt": 8113.99,

"currentDebt": 361.99,

"currency": "BYN",

"repaymentDate": "16.01.2023",

"rate": 22.8,

"iban": "AABBCCCDDDDEEEEEEEEEEEEEEEE",

"possibleRepayment": true,

"isClosed": false,

"customer\_type" : "LEGAL/PHYSIC"

}

}

{

"header": {

"message\_type": "currency-rate"

},

"payload": {

"startDt": "2024-01-03T13:56:51.604498616+03:00",

"exchangeRates": [

{

"buyRate": 3.33,

"sellRate": 3.43,

"srcCurr": "EUR",

"reqCurr": "BYN"

},

{

"buyRate": 3.05,

"sellRate": 3.15,

"srcCurr": "USD",

"reqCurr": "BYN"

},

{

"id": 5,

"buyRate": 1.075,

"sellRate": 1.1,

"srcCurr": "EUR",

"reqCurr": "USD"

}

]

}

}

{

"header": {

"message\_type": "customer"

},

"payload": {

"customer\_id": "1a72a05f-4b8f-43c5-a889-1ebc6d9dc729",

"customer\_type" : "LEGAL/PHYSIC",

"unp": "Только для LEGAL",

"register\_date": "dd.MM.yyyy",

"email": "example@email.com",

"phoneCode": "37529",

"phoneNumber": "1112233",

"customer\_fullname": "Иванов Иван Иванович"

}

}

{

"header": {

"message\_type": "deposit\_info"

},

"payload": {

"customer\_id": "1a72a05f-4b8f-43c5-a889-1ebc6d9dc729",

"customer\_type" : "LEGAL/PHYSIC",

"acc\_info": {

"acc\_iban": "AABBCCCDDDDEEEEEEEEEEEEEEEE",

"acc\_open\_date": "dd.MM.yyyy",

"curr\_amount": 3000.00,

"curr\_amount\_currency": "BYN"

},

"dep\_info": {

"rate": 14.50,

"term\_val": 24,

"term\_scale": "M/D",

"exp\_date": "dd.MM.yyyy",

"dep\_type": "REVOCABLE/IRREVOCABLE",

"auto\_renew": true

}

}

}

Требования:

1. Использовать Spring Boot 3.x, Java 17, Gradle и PostgreSQL.

2. Разработать API согласно подходам REST (UI не надо):

* CRUD для работы со счетами
* CRUD для работы с картами
* CRUD для работы с депозитами
* CRUD для работы с кредитами
* Просмотр всех сущностей клиента по его идентификатору

2.1. В ответе со списком счетов должны отображаться прикреплённые к ним карты (связаны по iban)

2.2 Вместе с балансом карты нужно отобразить эквиваленты в известных валютах (действующий курс приходит в сообщении currency-rate)

3. Разместить проекты (или проект, если решили работать в рамках нескольких модулей в одном репозитории) в любом из публичных git-репозиториев (Bitbucket, github, gitlab)

4. Код должен быть легко читаемый и понятный, с использованием паттернов проектирования

5. Реализовать на основе Spring @Profile (e.g. test & prod) подключение к базам данных.

6. Подключить liquibase:

- при запуске сервиса накатываются скрипты на рабочую БД (генерируются необходимые таблицы)

- при запуске тестов должен подхватываться скрипт по генерации необходимых таблиц + накатить данные по заполнению таблиц

7. Создать реализацию кэша, для хранения сущностей. Реализовать два алгоритма LRU и LFU. Алгоритм и максимальный размер коллекции должны читаться из файла application.yml. Алгоритм работы с кешем:

• GET - ищем в кеше и если там данных нет, то достаем объект из dao, сохранить в кэш и возвращаем

• POST - сохраняем в dao и потом сохраняем в кеше

• DELETE - удаляем из dao и потом удаляем в кеша

• PUT - обновление/вставка в dao и потом обновление/вставка в кеше.

8. Код должен быть покрыт юнит-тестами (80%)

9. Реализовать логирование запрос-ответ в аспектном стиле (для слоя Controlles), а также логирование по уровням в отдельных слоях приложения, используя logback

10. Предусмотреть обработку исключений и интерпретацию их согласно REST (см.<https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc>)

11. Все настройки должны быть вынесены в \*.yml

12. Использовать Spring REST Docs или другие средства автоматического документирования (например asciidoctor<https://asciidoctor.org/docs/asciidoctor-gradle-plugin/> и т.д) и/или Swagger (OpenAPI 3.0)

13. Использовать testcontainers в тестах на persistence layer (для БД)

14. Написать интеграционные тесты

15. Использовать WireMock в тестах для слоя clients

16. Использовать Docker (написать Dockerfile – для spring boot приложения, docker-compose.yml для поднятия БД и приложения в контейнерах и настроить взаимодействие между ними)

17\* Подключить кэш провайдер Redis (в docker) (в случае реализации, использовать @Profile для переключения между LRU/LFU и Redis)

18\* Spring Security:

* + Создавать новые сущности (карты, кредиты) можно только администратору.
  + Создавать и менять (переименовывать) сущности (счета, депозиты) может сам пользователь. Предусмотреть, что он может это делать только для сущностей, привязанных к нему.
  + Просматривать сущности пользователь может только свои. Администратор может запросить по любому пользователю.
  + Удалять сущности может только super\_user. Предусмотреть механизм soft delete.

19. \* Настроить Spring Cloud Config (вынести в отдельный сервис и настроить разрабатываемый сервис на получение их в зависимости от профиля)

1. \* Реализацию логирования п.10 и обработку исключений вынести в отдельные spring-boot-starter-ы.