**Задание**

Написать микросервисное приложение «Банк» с использованием Spring Boot и паттернов микросервисной архитектуры.

Рекомендуем вам создать HTML-страницы самостоятельно, чтобы получить полноценную практику работы с приложением. Но если у вас недостаточно времени, мы подготовили [архив с исходными файлами с HTML-страницами](https://code.s3.yandex.net/middle-java/s9-templates.zip?etag=dde4690e2b8107162a2b8d7981c27a94). В архиве две папки:

1. «Только html-шаблоны» — это только HTML с "...", где вам нужно самостоятельно подменить пути с ручками эндпоинтов и заполнить соответствующие атрибуты HTML-тегов.
2. «html-шаблоны» — это HTML-страницы с путями до ручек эндпоинтов, атрибутами HTML-тегов и т. д. (более простой вариант, но тогда вы завяжетесь на те эндпоинты и модели, которые уже прописаны).

В каждой папке есть readme.txt, где написано, что необходимо доделать в шаблоне, а также даны скриншоты страниц в браузере (чтобы понимать, как они должны выглядеть).

**Требования к приложению**

1. Микросервисы приложения написаны на Spring Boot.
2. Код приложения хранится в Git-репозитории на GitHub.
3. Микросервисы приложения собираются с помощью Maven или Gradle.
4. Есть возможность собирать сразу все микросервисы одной командой (например, с помощью parent POM и использования мультимодульных проектов).
5. Код приложения написан на Java 21.
6. Web UI приложения может использовать Spring Web MVC или Spring WebFlux.
7. Приложение может использовать Spring Data JDBC, Spring Data R2DBC или Spring Data JPA и Hibernate ORM под капотом для доступа к данным в базе данных.
8. Базы данных для микросервисов приложения должны быть персистентными (например, PostgreSQL). Можно обойтись одной базой данных, но разграничить микросервисы на уровне схем, чтобы реализовать паттерн Database per Service.
9. В качестве Service Discovery и Gateway API можно применять Consul, Eureka или самописный сервис, реализующий паттерны Service Discovery и Gateway API, с использованием проекта Spring Cloud Gateway.
10. Микросервисы должны выполнять запросы в другие микросервисы через Gateway API и регистрироваться в Service Discovery.
11. В качестве Externalized/Distributed Config можно применять Consul, ZooKeeper или самописный сервис, реализующий паттерн Externalized/Distributed Config, с использованием проекта Spring Cloud Config.
12. Микросервисы должны получать общие настройки из Externalized/Distributed Config.
13. В качестве сервера авторизации OAuth 2.0 можно использовать Keycloak, любой другой, который можно установить локально, или самописный сервер авторизации OAuth 2.0 с использованием проекта Spring Security OAuth.
14. В микросервисах авторизация должна осуществляться на сервере OAuth 2.0, чтобы можно было получать Access Token (JWT) по client credentials для выполнения запросов в другие микросервисы.
15. Микросервисы должны уметь выполнять ретраи запросов, а также иметь фоллбэки на эндпоинты заглушки (это могут быть Ribbon и Hystrix, Resilence4j и т. д. на ваш выбор).
16. Микросервисы приложения можно собрать из исходников и запустить локально.
17. Микросервисы приложения должны быть упакованы в Executable JAR и запускаться в Tomcat, Jetty или Netty.
18. Приложение покрыто тестами (юнит, интеграционными, контрактными) с использованием JUnit 5, TestContext Framework, Spring Boot Test, кеширования контекстов и Spring Cloud Contract.
19. Микросервисы должны быть написаны и протестированы с использованием паттернов микросервисной архитектуры приложения.
20. Executable JAR микросервисов должны быть упакованы в Docker-контейнеры с открытым веб-портом для доступа из браузера или межсервисного взаимодействия, чтобы реализовать паттерн Single Service per Host.
21. Запускать микросервисы можно все вместе с помощью Docker Compose или по одному вручную с помощью команд Docker.

**Функциональность**

1. Микросервисное приложение «Банк» — это приложение с веб-интерфейсом, которое позволяет пользователю (клиенту банка):
   * регистрироваться в системе по логину и паролю (заводить аккаунт);
   * добавлять счета в различных валютах;
   * класть виртуальные деньги на счёт и снимать их;
   * переводить деньги между своими счетами с учётом конвертации в различные валюты;
   * переводить деньги на другой счёт с учётом конвертации в различные валюты.
2. Приложение состоит из следующих микросервисов:
   * фронта (Front UI);
   * сервиса аккаунтов (Accounts);
   * сервиса обналичивания денег (Cash);
   * сервиса перевода денег между счетами одного или двух аккаунтов (Transfer);
   * сервиса конвертации валют (Exchange);
   * сервиса генерации курсов валют (Exchange Generator);
   * сервиса блокировки подозрительных операций (Blocker);
   * сервиса уведомлений (Notifications).

**Фронт (Front UI)**

Фронт (Front UI) — это веб-приложение с клиентским HTML-интерфейсом. Предоставляет следующие HTML-страницы:

1. Cтраницу ввода логина/пароля.

Можно предусмотреть аутентификацию remember-me. Можно использовать дефолтную страницу, которую предоставляет Spring Security.

Страница содержит:

* + поле ввода логина;
  + поле ввода пароля;
  + кнопку «Войти», при нажатии на которую происходит аутентификация/авторизация пользователя и редирект на главную страницу.

1. Страницу выхода из приложения (разлогинивания).

Можно использовать дефолтную страницу, которую предоставляет Spring Security.

Страница содержит ссылку, при нажатии на которую:

* + пользователь разлогинивается;
  + происходит редирект на страницу ввода логина/пароля.

1. Страницу регистрации пользователя в приложении (нового аккаунта).

Страница содержит:

* + поля ввода фамилии, имени, почты (если планируется отправлять уведомления на почту), даты рождения;
  + поля ввода логина и пароля;
  + кнопку «Зарегистрироваться», при нажатии на которую происходит проверка введённых данных и создаётся новый аккаунт с автоматической аутентификацией/авторизацией пользователя и редиректом на главную страницу (должна быть предусмотрена валидация: все поля заполнены, возраст старше 18 лет).

1. Главную страницу, которая доступна только после успешной аутентификации/авторизации пользователя.

Страница содержит:

* + блок настроек аккаунта;
  + блок внесения и снятия виртуальных денег;
  + блок перевода денег между своими счетами;
  + блок перевода денег на счёт другого аккаунта;
  + блок курсов валют.

**Блок настроек аккаунта**

Состоит из:

* логина пользователя без возможности редактирования, но с возможностью удаления (если есть хотя бы один ненулевой счёт, то должна появляться ошибка);
* поля ввода нового пароля и кнопки «Изменить пароль», при нажатии на которую меняется пароль пользователя (пароль должен быть непустой);
* фамилии, имени, почты (если предполагается отправлять уведомления на почту) и даты рождения с возможностью их редактирования (должна быть предусмотрена валидация: все поля заполнены, возраст старше 18 лет);
* списка счетов пользователя с возможностью их добавления, редактирования и удаления (у пользователя может быть не более одного счёта в определённой валюте);
* ссылки на страницу разлогинивания.

**Блок внесения и снятия виртуальных денег**

Состоит из:

* поля выбора счёта (обязательно);
* поля ввода суммы снятия (обязательно);
* кнопок «Положить» и «Снять» (если сумма, которую нужно снять, больше суммы на счёте, то должна появляться ошибка).

**Блок перевода денег между своими счетами**

Состоит из:

* поля выбора своего счёта для отправки денег (обязательно);
* поля выбора своего счёта для получения (обязательно);
* поля ввода суммы перевода (если сумма больше суммы на счёте отправления, то должна появляться ошибка);
* кнопки, при нажатии на которую осуществляется перевод денег.

**Блок перевода денег на счёт другого аккаунта**

Состоит из:

* поля выбора своего счёта для отправки денег (обязательно);
* поля выбора счёта получателя (обязательно, с поиском по аккаунту);
* поля ввода суммы перевода (если сумма больше суммы на счёте отправления, то должна появляться ошибка);
* кнопки, при нажатии на которую осуществляется перевод денег.

**Блок курсов валют**

1. Первый столбец таблицы — валюта.
2. Второй столбец — покупка.
3. Третий столбец — продажа.

**Сервис аккаунтов (Accounts)**

Сервис аккаунтов хранит информацию о зарегистрированных аккаунтах и счетах в каждом из них (именно в нём хранятся логин/пароль, которые проверяются при аутентификации пользователя).

* Фронт выполняет REST-запросы (в формате JSON) из блока настроек аккаунта в сервис Accounts при получении данных аккаунта и данных о счетах, их редактировании, добавлении и удалении.
* В свою очередь, Accounts выполняет REST-запросы (в формате JSON) в Notifications.
* Также в Accounts приходят запросы при регистрации нового аккаунта (из формы регистрации).

**Сервис обналичивания денег (Cash)**

Сервис обналичивания денег осуществляет пополнение счёта или снятие денег со счёта.

* Фронт выполняет REST-запросы (в формате JSON) из блока внесения и снятия виртуальных денег в сервис Cash.
* В свою очередь, Cash выполняет REST-запросы (в формате JSON) в Accounts, Blocker и Notifications.

**Сервис перевода денег между счетами (Transfer)**

Сервис перевода денег между счетами осуществляет перевод денег между счетами одного пользователя и между счетами разных пользователей.

* Фронт выполняет REST-запросы (в формате JSON) из блока перевода денег между своими счетами и блока перевода денег на счёт другого аккаунта в сервис Transfer.
* В свою очередь, Transfer выполняет REST-запросы (в формате JSON) в Accounts, Exchange, Blocker и Notifications.

**Сервис конвертации валют (Exchange)**

Сервис конвертации валют хранит информацию о конвертации валюты при её покупке/продаже (базовой валютой считается RUB, её конвертация при продаже/покупке равна 1).

Фронт выполняет REST-запросы (в формате JSON) из блока курсов валют в сервис Exchange для получения информации о курсах валют.

**Сервис генерации курсов валют (Exchange Generator)**

Сервис генерации курсов валют каждую секунду по расписанию генерирует курсы валют (алгоритм на ваш выбор: Round Robin из файла, БД, случайные значения и т. д.) и выполняет REST-запросы (в формате JSON) в сервис Exchange.

Валют должно быть не меньше трёх: RUB, USD, CNY. Базовой валютой считается RUB, то есть при конвертации, например, из USD в CNY сначала USD конвертируется в RUB, а потом RUB в CNY.

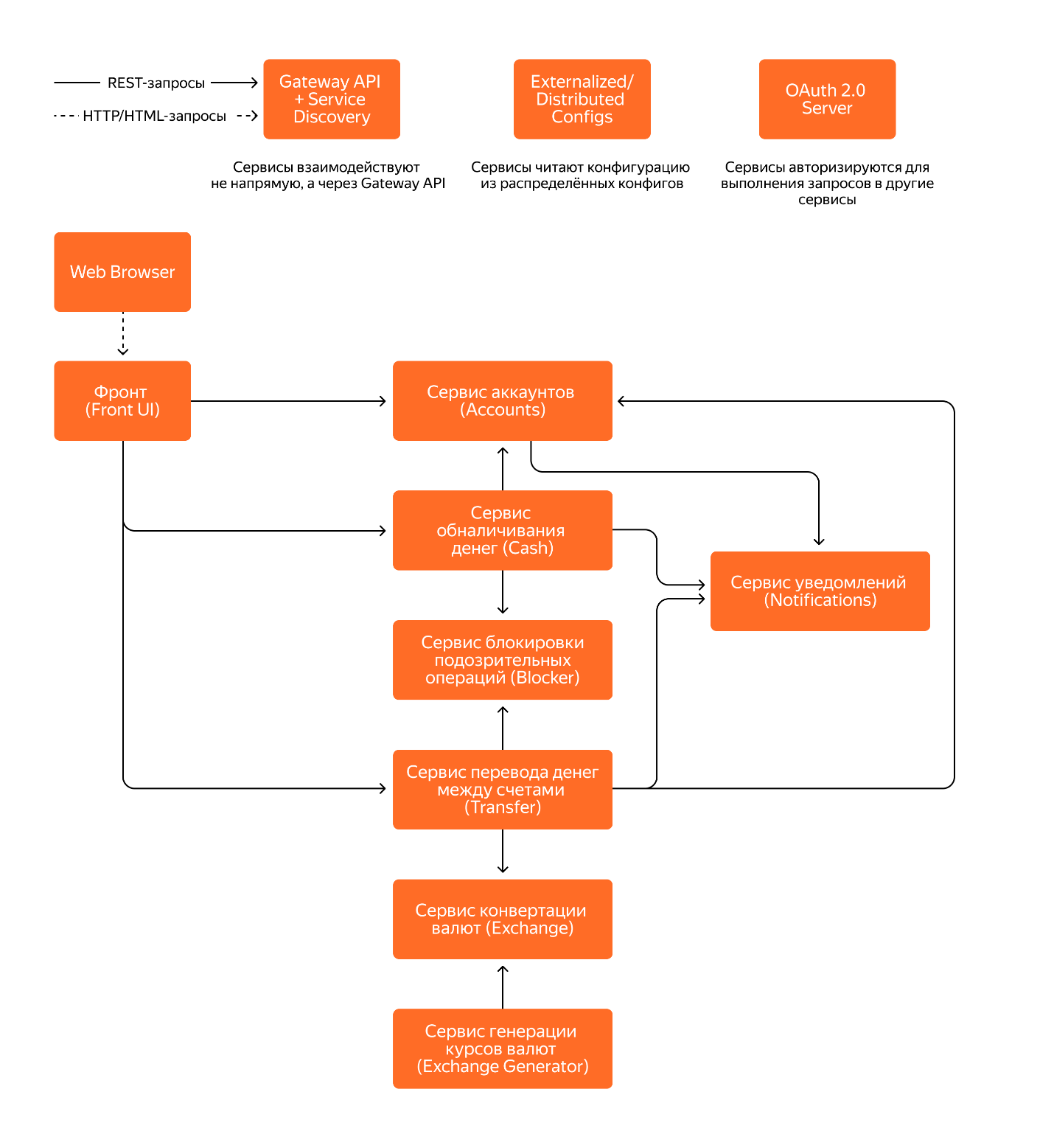
**Сервис блокировки подозрительных операций (Blocker)**

Сервис блокировки подозрительных операций отслеживает подозрительные операции (алгоритм на ваш выбор: случайно, в зависимости от текущего времени и т. д.).

**Сервис уведомлений (Notifications)**

Сервис уведомлений отправляет уведомления (на ваш выбор: на почту, выдаёт Alert и т. д.) пользователю о выполненном действии: переводе денег, пополнении счёта, снятии денег со счёта и т. д.

Итоговая схема взаимодействия сервисов выглядит так:



1. Микросервисы должны аутентифицироваться/авторизовываться на сервере авторизации по client credentials, чтобы выполнять запросы в другие микросервисы. У одного микросервиса может не быть доступа к выполнению запросов в другой микросервис (например, микросервис Exchange не может выполнять запросы в Accounts).
2. Аутентификация/авторизация пользователей должны быть по логину/паролю. У пользователя есть доступ к информации только о своём аккаунте и счетах.

**Как работать над заданием**

1. Изучить описание задания (веб-страницы, функциональность, типы микросервисов, их взаимодействие).
2. Спроектировать и реализовать приложение в соответствии с требованиями проекта.
3. Стараться использовать микрокоммиты при разработке приложения.
4. В финальном коммите проставить тег v1.0 и сделать пуш на GitHub.

**Шаги по разработке приложения**

1. Для проекта создать новый репозиторий в своём профиле на GitHub (например, с названием bankapp). Профиль должен быть **публичным**, чтобы ревьюер мог с ним работать.
2. Клонировать репозиторий к себе в локальный Git-репозиторий.
3. Спроектировать микросервисы в соответствии с требованиями и функциональностью проекта, а также с паттернами микросервисных приложений.
4. В каждом микросервисе спроектировать слои согласно паттернам разработки корпоративных приложений (view только для Front UI, controller/REST-controller, service, model, DAO).
5. Спроектировать классы/подмодули в каждом слое согласно принципам и паттернам разработки SOLID, YAGNI и т. д.
6. Создать микросервисы с нуля с использованием инструментов Spring Boot (например, start.spring.io).
7. Объединить все микросервисы под одним родительским проектом (мультипроектом, в котором каждый микросервис — это подпроект).
8. Создать проект для этого мультипроекта в среде разработки (например, в IntelliJ Idea).
9. Инициализировать проект Git.
10. Выбрать систему сборки (Maven/Gradle).
11. Установить одну СУБД с разграничением схем (для каждого микросервиса своя схема) или по одной СУБД для каждого микросервиса, которому она нужна (можно в контейнере).
12. Установить и настроить Consul, если используется в проекте.
13. Создать Spring Boot-приложение с Eureka Server и настроить его, если используется в проекте.
14. Создать и настроить Spring Boot-приложение с Gateway API, Service Discovery и Externalized/Distributed Configs, если используются самописные решения.
15. Установить и настроить OAuth 2.0 Server (например, Keycloak), если он используется в проекте. Сгенерировать Spring Boot-проект со Spring Security OAuth Server и настроить его, если используется самописный сервер авторизации OAuth 2.0.
16. Добавить зависимости на Spring Boot, зависимости и стартеры Spring WebMVC, Spring WebFlux, Spring Data JDBC, JPA, R2DBC, Spring Cloud проекты и т. д.
17. Написать классы слоя модели, DAO, сервисов, контроллеров, view (шаблонизатор Thymeleaf), REST-клиентов в каждом из микросервисов.
18. Написать модульные, интеграционные и контрактные тесты на микросервисы и взаимодействие между ними.
19. Написать сборку Executable JAR для каждого из микросервисов.
20. Написать Dockerfile для развёртывания и запуска каждого приложения в Docker. Можно написать Docker Compose для развёртывания и запуска всего мультипроекта.
21. Написать README.md (как собирать и запускать приложение в среде разработки, локально и в Docker).
22. В процессе разработки приложения делать микрокоммиты в Git, дописывать файлы описания сборки Maven/Gradle, накатки схемы БД и т. д.
23. В финальном коммите проставить тег v1.0 и сделать пуш на GitHub.

**Что проверит ревьюер**

* Правильность описания мультипроекта и подпроектов для каждого микросервиса в pom.xml или build.gradle.
* Структуру, схему взаимодействия микросервисов, права доступа, роли при взаимодействии микросервисов в соответствии с паттернами разработки микросервисных приложений: Circuit Breaker, Service Discovery, Gateway API, RPI, Transactional Outbox, Access Token, UI Composition, Contract Testing, Externalized/Distributed Configs и т. д.
* Структуру каждого микросервиса согласно модулям/частям микросервиса.
* Структуру каждого микросервиса согласно слоям приложения и правильное расположение слоёв и взаимодействие между ними.
* Структуру и взаимодействие классов согласно принципам и паттернам разработки корпоративных приложений.
* Наименование классов, методов, полей и других элементов в соответствии с их назначением.
* Правильность использования Spring Boot для разработки веб-приложения и RESTful-сервисов.
* Правильность реализации авторизации по логину/паролю для пользователя и OAuth 2.0 для микросервисов при взаимодействии между собой.
* Правильность проектирования структур БД для хранения данных.
* Покрытие модульными, интеграционными и контрактными тестами в соответствии с заданием.
* Правильность написания Dockerfile для развёртывания и запуска микросервисов в Docker-контейнере, а также файла Docker Compose.
* Подробность и понятность описания проекта в README.md, форматирование.
* Использование микрокоммитов при разработке приложения.
* Правильное использование rebase/merge-слияний в истории проекта.
* Зелёные прогоны тестов.

**Что не нужно делать (будет в следующих спринтах)**

* Настраивать репликации микросервисов и Load Balancer.
* Использовать шины данных, JMS, Kafka для межсервисного взаимодействия и поставки логов.
* Агрегировать, хранить и анализировать логи (ELK-стек).
* Настраивать Health Check, мониторинг микросервисов, аудит.
* Настраивать распределённые трассировки.
* Развёртывать микросервисы в Kubernetes.
* Использовать Jenkins CI/CD или аналоги.