Geekbrains

**«Банковский сервис на Java Spring Boot» — фулстек-разработка от бизнес-требований до MVP**

IT-специалист:

Программирование

Титов Е.В.

Москва

2024

**СОДЕРЖАНИЕ**

Введение.......................................................................................................................3

Глава 1. Теоретические основы разработки программного обеспечения на микросервисной архитектуре.....................................................................................7

1.1. Понятие микросервисной архитектуры..............................................................7

1.2. Особенности разработки на микросервисной архитектуре…..........................9

Глава 2. Разработка требований к банковскому сервису………………...…..…..12

2.1. Бизнес-требования к системе............................................................................12

2.2. Сбор и анализ требований.................................................................................15

Глава 3. Реализация банковского сервиса на Java Spring Boot.............................17

3.1. Выбор используемого стека технологий..........................................................17

3.2. Архитектура проекта..........................................................................................19

3.3 Архитектура сервисов…………………………………….................................21

Заключение.................................................................................................................24

Список использованной литературы.......................................................................25

Приложение................................................................................................................26

**ВВЕДЕНИЕ**

В современном мире информационные технологии играют ключевую роль в развитии различных отраслей экономики, банковская сфера не является исключением. С ростом конкуренции и увеличением требований клиентов к качеству услуг, банки вынуждены постоянно совершенствовать свои системы и внедрять новые технологии для обеспечения более высокого уровня обслуживания.

Банковское программное обеспечение (ПО) является критически важным для функционирования банков и финансовых учреждений по нескольким причинам:

*Безопасность данных*: Банковское ПО отвечает за обработку и хранение конфиденциальной информации о клиентах, включая финансовые данные, личные сведения и историю транзакций. Утечка или несанкционированный доступ к этой информации могут привести к серьезным последствиям, таким как мошенничество, потеря доверия клиентов и юридические проблемы.

*Управление рисками*: Банки используют ПО для управления рисками, связанными с финансовыми операциями. Это включает в себя оценку кредитоспособности клиентов, управление ликвидностью и рисками ликвидности, а также контроль за соблюдением регуляторных требований.

*Операционные процессы*: Банковские операции, такие как обработка платежей, выдача кредитов, открытие счетов и другие, зависят от надежного и эффективного ПО. Сбои в работе ПО могут привести к задержкам в обслуживании клиентов, потере доходов и снижению репутации банка.

*Соответствие стандартам и нормам:* Банки должны соблюдать строгие стандарты и нормы, установленные регуляторами, такими как Центральный банк России. Банковское ПО должно быть разработано и настроено в соответствии с этими требованиями.

*Масштабируемость и гибкость:* В условиях быстро меняющегося рынка и растущих потребностей клиентов банки должны иметь возможность быстро адаптироваться к новым требованиям и условиям. Банковское ПО должно обеспечивать масштабируемость и гибкость для удовлетворения этих потребностей.

*Интеграция с другими системами:* Банковское ПО часто интегрируется с другими финансовыми системами, такими как системы учета, управления активами и т.д. Сбой в интеграции может привести к нарушению работы всей финансовой инфраструктуры.

*Защита от киберугроз:* Банки являются привлекательными целями для киберпреступников из-за ценности их данных. Банковское ПО должно включать в себя меры безопасности для защиты от кибератак, таких как шифрование данных, аутентификация пользователей и антивирусная защита.

*Автоматизация процессов:* Банковское ПО автоматизирует рутинные задачи, такие как расчет процентов, формирование отчетов и анализ данных. Это позволяет банкам повысить эффективность и снизить вероятность ошибок, связанных с человеческим фактором.

*Поддержка клиентов:* Банковское ПО обеспечивает поддержку клиентов через онлайн-сервисы, мобильные приложения и другие каналы. Сбои в этой системе могут негативно повлиять на удовлетворенность клиентов и их лояльность.

В целом, банковское ПО играет ключевую роль в обеспечении безопасности, эффективности и надежности банковских операций, что делает его критически важным (Mission Critical System).

Margaret Rouse: «A mission critical system is a system that is essential to the survival of a business or organization. When a mission critical system fails or is interrupted, business operations are significantly impacted[[1]](#footnote-1)».

Одним из наиболее перспективных направлений развития банковских систем является разработка микросервисов на основе Java Spring Boot. Микросервисы представляют собой модульную архитектуру, которая позволяет разделить систему на небольшие независимые компоненты, что обеспечивает гибкость, масштабируемость и надежность, а также снижения time-to-market, однако имеет ряд «подводных камней».

**Актуальность данной темы** обусловлена следующими факторами:

Рост конкуренции в банковской сфере: В условиях жесткой конкуренции банки должны постоянно улучшать качество своих услуг и предлагать клиентам новые возможности. Разработка микросервисов позволяет быстро и эффективно внедрять изменения в систему, обеспечивая тем самым высокий уровень обслуживания.

Требования к безопасности и надежности: Банки обрабатывают большие объемы конфиденциальной информации, поэтому безопасность и надежность системы являются критически важными. Микросервисная архитектура позволяет обеспечить высокий уровень защиты данных и устойчивость к сбоям.

Масштабируемость и гибкость: Современные банковские системы должны быть способны быстро адаптироваться к изменяющимся требованиям рынка и новым технологиям. Микросервисы позволяют легко добавлять или изменять отдельные компоненты системы без необходимости перестройки всей архитектуры.

Интеграция с другими системами: Банковские системы часто интегрируются с различными внешними сервисами и платформами. Микросервисы обеспечивают более гибкую и эффективную интеграцию, позволяя легко подключать новые сервисы и адаптировать систему под изменяющиеся требования.

Перспективы развития: Микросервисная архитектура является перспективным направлением развития информационных технологий. Она позволяет создавать гибкие, масштабируемые и надежные системы, которые соответствуют современным требованиям банков.

**Целью данного исследования** является выявление лучших практик разработки микросервисов для создания банковского сервиса на Java Spring Boot, начиная от бизнес-требований и заканчивая созданием MVP (Minimum Viable Product).

Для достижения этой цели необходимо решить следующие задачи:

Изучить основные принципы и особенности микросервисной архитектуры.

Проанализировать существующие практики разработки микросервисов.

Разработать рекомендации по созданию микросервисов для банковского сервиса.

Создать MVP на основе разработанных рекомендаций.

Для разработки такой системы необходимо создание отдельных команд, каждая из которых будет развивать и поддерживать свой компонент (микросервис). Вот примерный состав команд:

Системный и/или бизнес-аналитик (зачастую данные роли объединяет один человек), разработчики, тех. лид (архитектор или разработчик уровня senior), тестировщики, devOps-инженер, админситартор.

Межкомандные роли: Корпоративный архитектор, системные администраторы и devOps-инженеры (зачастую работают за пределами одной команды), product owner, специалисты информационной безопасности.

Product owner — это специалист, который отвечает за управление продуктом в процессе разработки. Product owner представляет интересы всех заинтересованных сторон и несёт ответственность за успех продукта на рынке.

Основные обязанности product owner:

1. определение требований к продукту;
2. приоритизация задач;
3. взаимодействие с командой разработчиков;
4. анализ рынка и конкурентов;
5. контроль выполнения задач и сроков;
6. отчётность о проделанной работе.

Структуры команд могут варьироваться в зависимости от сложности проекта и требований заказчика. Однако в любом случае команда/команды должна быть хорошо скоординированы и работать над проектом в тесном сотрудничестве.

**ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА МИКРОСЕРВИСНОЙ АРХИТЕКТУРЕ**

* 1. **Понятие микросервисной архитектуры**

В современном мире информационных технологий и постоянно растущих требований к гибкости, масштабируемости и надёжности программных систем, микросервисная архитектура становится всё более популярной. Она представляет собой подход к разработке программного обеспечения, при котором система разбивается на небольшие независимые сервисы, каждый из которых выполняет свою конкретную функцию.

Микросервисная архитектура — это подход к проектированию и разработке программного обеспечения, основанный на использовании небольших, независимых сервисов, которые взаимодействуют друг с другом через хорошо определённые интерфейсы. Эти сервисы могут быть написаны на разных языках программирования, использовать разные технологии и базы данных, но при этом они должны обеспечивать согласованность и целостность системы в целом.

MSA[[2]](#footnote-2) — принципиальная организация распределенной системы на основе микросервисов и их взаимодействия друг с другом и со средой по сети, а также принципов, направляющих проектирование архитектуры, её создание и эволюцию.

Свойства микросервиса:

1. Он небольшой.
2. Он независимый.
3. Он строится вокруг бизнес-потребности и использует ограниченный контекст (Bounded Context).
4. Он взаимодействует с другими микросервисами по сети на основе паттерна Smart endpoints and dumb pipes.
5. Его распределенная суть обязывает использовать подход Design for failure.
6. Централизация ограничена сверху на минимуме.
7. Процессы его разработки и поддержки требуют автоматизации.
8. Его развитие итерационное.

Основные принципы микросервисной архитектуры включают:

*Независимость сервисов*: каждый сервис должен быть независимым от других сервисов и иметь свой собственный жизненный цикл. Это позволяет легко масштабировать и обновлять отдельные сервисы без влияния на всю систему.

*Слабая связанность*: между сервисами должна быть слабая связанность, что означает, что они взаимодействуют только через чётко определённые API. Это обеспечивает гибкость и возможность изменения одного сервиса без необходимости переписывать другие.

*Децентрализованное управление*: каждый сервис имеет своё собственное управление и может быть развёрнут, обновлён или масштабирован независимо от других.

*Автоматизация развёртывания и управления*: использование инструментов автоматизации для развёртывания, обновления и управления сервисами.

Преимущества микросервисной архитектуры заключаются в следующем:

*Гибкость и масштабируемость:* благодаря независимости сервисов, можно легко добавлять, удалять или изменять их в соответствии с требованиями бизнеса.

*Надёжность:* если один сервис выходит из строя, остальные продолжают работать, обеспечивая непрерывность работы системы.

*Простота разработки и тестирования:* небольшие сервисы легче разрабатывать, тестировать и поддерживать.

*Возможность использования различных технологий*: каждый сервис может использовать свои собственные технологии, что позволяет выбирать наиболее подходящие решения для каждой задачи.

Однако микросервисная архитектура также имеет некоторые недостатки:

*Сложность интеграции:* взаимодействие между сервисами может потребовать дополнительных усилий по разработке и тестированию.

*Увеличение количества компонентов*: большее количество сервисов может усложнить управление системой.

*Необходимость в инструментах автоматизации*: для эффективного управления микросервисной архитектурой требуются инструменты автоматизации.

*Большая нагрузка на сеть:* в сравнении с монолитом получаем: то, что раньше вызывалось напрямую в коде, теперь требует интеграции.

*Сложность проектирования и выделения доменов микросервисов:* проектирование и выделение доменов микросервисов требует глубокого понимания бизнес-процессов, технических аспектов и принципов разработки. Этот процесс может занять значительное время и ресурсы, но он позволяет создать гибкую и масштабируемую систему, способную адаптироваться к изменяющимся требованиям и условиям.

Кроме того, микросервисная архитектура значительно дороже в цене:

1. Требуется больше специалистов в идеале каждая команда должна разрабатывать и поддерживать один микросервис.
2. Необходима более развитая инфраструктура с точки зрения железа и сетей.
3. Большее количество сред для разработки и тестирования в идеале для каждого микросервиса должны быть дев, тест, предпродуктивная среды, в которых все остальные сервисы соответствуют продуктивной среде, что также требует дополнительных физических ресурсов (железа) и специалистов.
   1. **Особенности разработки на микросервисной архитектуре**

Разработка на микросервисной архитектуре имеет ряд особенностей, которые необходимо учитывать.

Вот некоторые ключевые аспекты, которые необходимо учитывать при проектировании и выделении доменов:

*Анализ бизнес-требований:*

Прежде чем приступать к проектированию, необходимо провести анализ бизнес-требований и определить основные функции и процессы, которые должны быть реализованы в системе. Это поможет определить, какие сервисы необходимы и как они должны взаимодействовать друг с другом.

*Определение границ сервисов:*

Важно определить границы каждого сервиса и его ответственность. Каждый сервис должен выполнять одну или несколько конкретных функций и не зависеть от других сервисов.

*Выбор архитектуры:*

Необходимо выбрать архитектуру, которая будет наиболее подходящей для реализации системы. Существует несколько популярных архитектур, таких как RESTful API, GraphQL и gRPC. Выбор зависит от требований к производительности, безопасности и масштабируемости.

*Обеспечение взаимодействия между сервисами:*

Необходимо разработать механизмы взаимодействия между сервисами, такие как API-интерфейсы, события и очереди сообщений. Эти механизмы должны обеспечивать надёжное и безопасное взаимодействие между сервисами.

*Тестирование и развёртывание:*

После проектирования необходимо провести тестирование каждого сервиса отдельно и всей системы в целом. Также необходимо разработать стратегию развёртывания системы, включая выбор облачной платформы, управление конфигурацией и мониторинг.

*Мониторинг и оптимизация:*

В процессе эксплуатации системы необходимо проводить мониторинг производительности и доступности сервисов. На основе полученных данных можно оптимизировать систему, например, путём добавления новых сервисов или изменения существующих.

*Масштабирование:*

При увеличении нагрузки на систему необходимо обеспечить её масштабирование. Для этого можно использовать горизонтальное масштабирование, то есть добавление новых экземпляров сервисов, или вертикальное масштабирование, то есть увеличение ресурсов каждого экземпляра.

*Безопасность:*

Необходимо обеспечить безопасность системы, используя такие меры, как аутентификация, авторизация, шифрование данных и защита от атак.

*Управление версиями:*

Необходимо управлять версиями сервисов и их зависимостей, чтобы обеспечить совместимость и избежать проблем при обновлении системы.

**ГЛАВА 2. РАЗРАБОТКА ТРЕБОВАНИЙ К БАНКОВСКОМУ СЕРВИСУ**

**2.1. Бизнес-требования к системе**

Бизнес-требование[[3]](#footnote-3) – высокоуровневая бизнес цель организации или заказчиков системы.

Бизнес-требования определяют цели и задачи, которые должна решать система, а также её основные функции и характеристики. В данном разделе будут рассмотрены бизнес-требования, предъявляемые к разрабатываемой системе банковского сервиса.

Вигерс в своей книге «Разработка требований к программному обеспечению» выделяет следующие уровни требований:

Бизнес-требования (Business Requirements) — это высокоуровневые цели организации или клиента, которые должны быть достигнуты с помощью системы. Они определяют, что система должна делать для достижения бизнес-целей.

Пользовательские требования (User Requirements) — описывают, как пользователи будут взаимодействовать с системой и какие функции она должна предоставлять. Эти требования обычно формулируются на основе анализа потребностей пользователей.

Функциональные требования (Functional Requirements) — определяют конкретные функции, которые должна выполнять система. Они могут включать в себя такие аспекты, как ввод данных, обработка информации, вывод результатов и т.д.

Нефункциональные требования (Non-functional Requirements) — касаются характеристик системы, таких как производительность, безопасность, надёжность, удобство использования и другие.

Системные требования (System Requirements) — включают в себя аппаратные и программные компоненты, необходимые для работы системы.

Требования к интерфейсу (Interface Requirements) — определяют взаимодействие между системой и пользователями, включая графический интерфейс пользователя (GUI), меню, формы ввода данных и т.п.

Требования безопасности (Security Requirements) — обеспечивают защиту системы от несанкционированного доступа, вирусов, хакерских атак и других угроз.

Юридические требования (Legal Requirements) — соответствуют законодательству и нормативным актам, регулирующим деятельность организации.

Типы требований к ПО

В зависимости от содержания и назначения, требования к ПО можно разделить на несколько типов:

Функциональные: описывают, какие задачи должна решать система, и какие действия должен выполнять пользователь. Например, возможность создания и редактирования документов, поиск информации, отправка сообщений и т. д.

Нефункциональные: определяют характеристики системы, не связанные с её функциональностью. К ним относятся требования к производительности, безопасности, надёжности, удобству использования, совместимости и др.

Технические: содержат информацию о технических аспектах разработки системы. Это может быть выбор платформы, технологий, инструментов и т. п.

Бизнес-правила: устанавливают правила и ограничения, связанные с бизнес-процессами организации. Например, правила расчёта налогов, условия предоставления скидок и т.д.

Ограничения: представляют собой факторы, которые необходимо учитывать при разработке системы.

Цели и задачи системы:

Повышение эффективности работы банка за счёт автоматизации процессов обслуживания клиентов.

Улучшение качества обслуживания клиентов путём предоставления им более удобных и быстрых способов взаимодействия с банком.

Снижение затрат на обслуживание клиентов за счёт оптимизации внутренних процессов банка.

Расширение спектра предоставляемых услуг и повышение их доступности для клиентов.

*Основные функции системы:*

Регистрация и авторизация клиентов: система должна обеспечивать возможность регистрации новых клиентов и их авторизации для доступа к сервисам банка.

Управление счетами: система должна предоставлять клиентам возможность управлять своими счетами, включая просмотр баланса, совершение операций по счёту и получение выписок (mvp включает только просмотр баланса).

Открытие нового банковского продукта: система должна предоставлять клиенту возможность открыть новый продукт.

В системе должен быть реализован бэк-офисный процесс введения реестра пластиковых карт, должна быть реализован интерфейс для управления и мониторинга данного процесса.

Безопасность и конфиденциальность: система должна обеспечивать высокий уровень безопасности данных клиентов и защиту от несанкционированного доступа.

Масштабируемость и гибкость: система должна иметь возможность масштабирования для удовлетворения растущих потребностей банка и адаптации к изменениям в законодательстве и технологиях.

Требования невошедшие в mvp проект:

Оплата товаров и услуг: система должна поддерживать различные способы оплаты, такие как банковские карты, электронные кошельки и другие платёжные инструменты.

Перевод средств: система должна позволять клиентам переводить средства между своими счетами в банке, а также на счета других банков.

Запрос и предоставление информации: система должна предоставлять клиентам информацию о своих счетах, услугах и продуктах банка, а также отвечать на их запросы.

Поддержка мобильных устройств: система должна быть доступна с мобильных устройств, таких как смартфоны и планшеты.

Интеграция с другими системами: система должна интегрироваться с внутренними системами банка для обмена данными и обеспечения согласованности информации.

Отчётность и аналитика: система должна генерировать отчёты о деятельности банка и предоставлять аналитические данные для принятия управленческих решений.

Эти требования являются основой для разработки системы банковского сервиса, которая будет соответствовать потребностям банка и его клиентов. Далее необходимо провести анализ существующих систем и определить, какие из них могут быть использованы в качестве основы для новой системы. Также необходимо разработать технические требования к системе, которые будут определять её архитектуру, компоненты и интерфейсы.

**2.2. Сбор и анализ требований**

В данном разделе описывается процесс сбора и анализа требований к разрабатываемому программному обеспечению (ПО). Этот этап является ключевым для успешного выполнения проекта, так как он определяет цели и задачи разработки, а также основные функции и характеристики будущего продукта.

Основные задачи:

* + Сбор информации от заинтересованных сторон (заказчик, пользователи, эксперты) о том, какие функции и возможности должны быть реализованы в ПО.
  + Анализ полученной информации и определение основных требований к системе.
  + Документирование требований в виде спецификации или технического задания.

Для сбора требований могут использоваться различные методы, такие как:

* + Интервью с заинтересованными сторонами (заказчики, пользователи, разработчики).
  + Анкетирование и опросы.
  + Наблюдение за работой пользователей.
  + Изучение существующих систем и документов.

Выбор метода зависит от специфики проекта и доступных ресурсов. Важно обеспечить участие всех заинтересованных сторон в процессе сбора требований, чтобы получить наиболее полную и точную информацию.

После сбора информации необходимо провести анализ требований, чтобы определить их полноту, непротиворечивость и выполнимость. Для этого можно использовать следующие методы:

* + Группировка требований по категориям (функциональные, нефункциональные).
  + Проверка на наличие противоречий между требованиями.
  + Оценка выполнимости требований с учетом ограничений проекта.
  + Результатом анализа является спецификация требований, которая будет использоваться на следующих этапах разработки ПО.

Результаты анализа требований должны быть документированы в виде спецификации требований или технического задания. В этом документе должны быть указаны все функциональные и нефункциональные требования к системе, а также ограничения и допущения проекта. Спецификация должна быть понятна и доступна всем участникам проекта. В качестве основного артефакта данного этапа подготовлен проект спецификации требований к банковскому сервису (см. Приложения: приложение №1).

Отмечается, что mvp не отвечает всем этим требованиям, а лишь представляет собой вариант экспериментального запуска пилотного проекта, что позволит бизнесу и ИТ-командам оценить правильность принятых решений, оттестировать процесс выпуска нового продукта и принять решения о дальнейшей реализации бизнес-задачи.

Кроме того, разработаны спецификации для АПИ сервисов   
(см. Приложения № 2 и 3): «Управление пластиковыми картами» и «Сервис управления продуктовым профилем клиента», а также пример технического задания на проект (сокращенный вариант).

**ГЛАВА 3. РЕАЛИЗАЦИЯ БАНКОВСКОГО СЕРВИСА   
НА JAVA SPRING BOOT**

**3.1. Выбор используемого стека технологий**

Для реализации банковского сервиса, необходимо выбрать подходящий стек технологий, который будет соответствовать требованиям проекта и обеспечивать необходимую функциональность.

Java Spring Boot — это фреймворк для создания веб-приложений на основе Java, который упрощает разработку и обеспечивает быструю настройку проекта. Он предоставляет готовые решения для многих задач, таких как управление зависимостями, конфигурация, тестирование, обеспечение безопасности, мониторинг ресурсов и т.д.

Выбор стека технологий зависит от требований проекта, его сложности, масштабируемости, безопасности и других факторов. Для реализации банковского сервиса можно использовать следующие технологии:

Java — основной язык программирования, используемый для разработки приложения.

Spring Boot — фреймворк, обеспечивающий быструю разработку и настройку приложения.

Hibernate — ORM-фреймворк для работы с базами данных встроен в Spring Data.

H2 — реляционная база данных, используемая для хранения данных приложения in memory (достаточно для первой итерации пилота).

Maven — инструмент для управления зависимостями и сборки проекта.

JUnit — фреймворк для тестирования кода.

Thymeleaf — шаблонизатор для генерации HTML-страниц.

Spring Security — модуль для обеспечения безопасности приложения.

Spring Data JPA – это интерфейсы, которые можно определять для получения доступа к данным. JPA-запросы создаются автоматически на основе имен методов.

OpenApi – для автоматической генерации API документации и удобства тестирования методов API.

Lombok (далее – просто Lombok) – это основанная на аннотациях библиотека Java, позволяющая сократить шаблонный код. В Lombok предлагаются различные аннотации, цель которых – заменить ненужный повторяющийся код

Spring Cloud — это модуль, который позволяет относительно быстро и легко создавать шаблоны в распределённых системах.

Среди возможностей Spring Cloud:

* + шаблоны управления конфигурацией;
  + шаблоны интеллектуальной маршрутизации;
  + шаблоны обнаружения сервисов;
  + одноразовые токены;
  + микропрокси и так далее.

Prometheus — набор инструментов с открытым исходным кодом, написанный на языке программирования Golang и представляющий собой полноценную систему мониторинга и оповещения.

Spring Cloud Gateway - предоставление простого, но эффективного способа подключения к API и решения для них сквозных задач, таких как безопасность, мониторинг / метрики и отказоустойчивость.

Grafana — свободная программная система визуализации данных, ориентированная на данные систем ИТ-мониторинга. Реализована как веб-приложение в стиле «приборных панелей» с диаграммами, графиками, таблицами, предупреждениями[[4]](#footnote-4).

Этот стек технологий обеспечивает необходимую функциональность для реализации банковского сервиса, включая обработку транзакций, аутентификацию пользователей, авторизацию доступа к данным и другие функции.

Также можно рассмотреть возможность использования дополнительных технологий, таких как Kafka для обработки событий, Redis для кэширования данных и другие, в зависимости от конкретных требований проекта.

Важно провести анализ требований и выбрать наиболее подходящий стек технологий для реализации проекта. Это позволит обеспечить высокую производительность, безопасность и удобство разработки и развертывания.

**3.2. Архитектура проекта**

Проект представляет собой набор сервисов, которые взаимодействуют друг с другом для обеспечения функциональности системы. В основе проекта лежит микросервисная архитектура, которая позволяет разделить систему на отдельные сервисы, каждый из которых выполняет свою функцию.

Сервис управления продуктовым профилем клиента

Этот сервис отвечает за управление информацией о продуктах, которые предлагаются клиенту. Он содержит информацию о ценах, характеристиках и наличии товаров. Сервис предоставляет API для других сервисов, чтобы они могли получить доступ к этой информации.

Сервис управления банковскими картами

Этот сервис управляет информацией о банковских картах клиентов. Он хранит данные о номерах карт, сроках действия и CVV-кодах. Сервис также предоставляет API для других сервисов, чтобы они могли выполнять операции с банковскими картами, поддержка возможности импорта файла в формате csv.

API Gateway

API Gateway является шлюзом между внешними клиентами и внутренними сервисами. Он обеспечивает безопасность и аутентификацию запросов, а также маршрутизацию запросов к соответствующим сервисам. API Gateway также может предоставлять дополнительные функции, такие как кэширование ответов и обработка ошибок.

Сервис Eruika

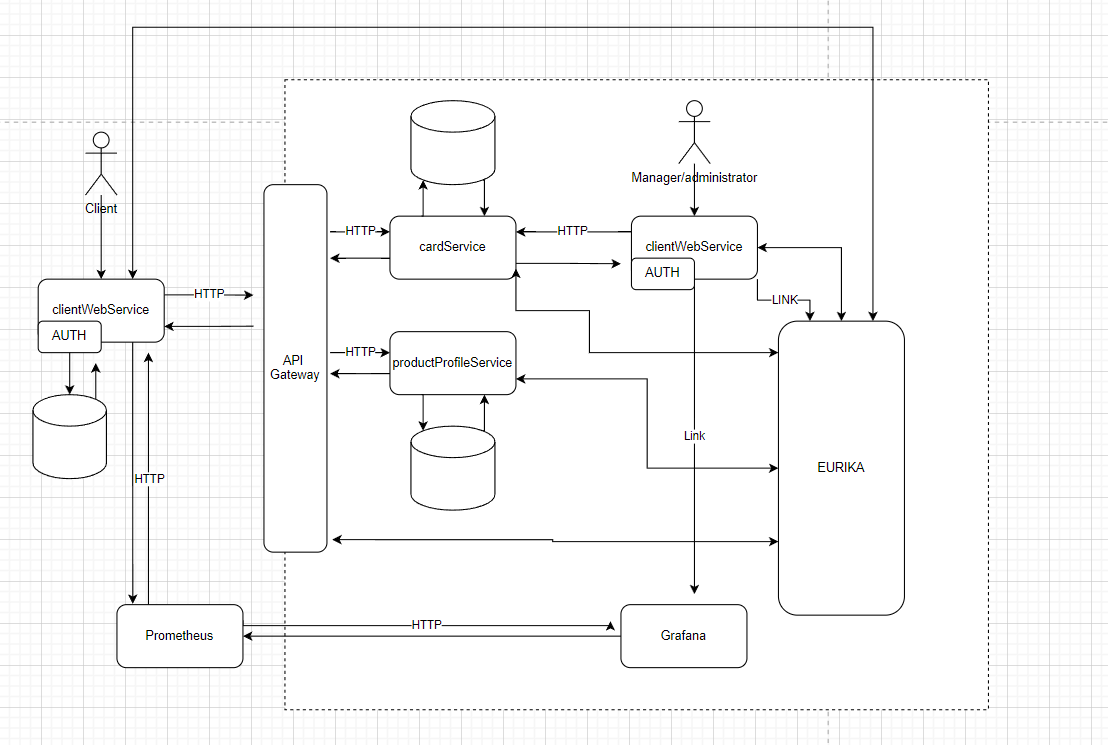
Сервис Eruika выполняет специфические функции, связанные с регистрацией и журналированием активности микросервисов, что в дальнейшем может использоваться оркестраторами при горизонтальном масштабировании отдельных сервисов.

Веб-сервис клиента

Веб-сервис клиента предоставляет интерфейс для взаимодействия с системой. Клиенты могут использовать веб-сервис для поиска продуктов, оформления заказов и выполнения других операций. Веб-сервис использует API других сервисов для получения необходимой информации.

Веб-сервис администратора

Веб-сервис администратора предоставляет интерфейс для управления системой. Администраторы могут использовать веб-сервис для настройки параметров системы, управления пользователями и выполнения других административных задач.



*рис. 1 Архитектура проекта*

**3.3 Архитектура сервисов**

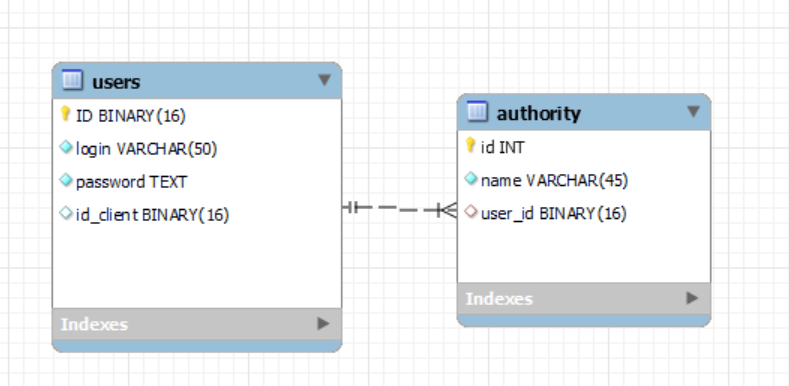
Все сервисы реализованы по паттерну MVC. Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») - схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер - таким образом, что модификация каждого компонента может осуществляться независимо[[5]](#footnote-5).

Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.

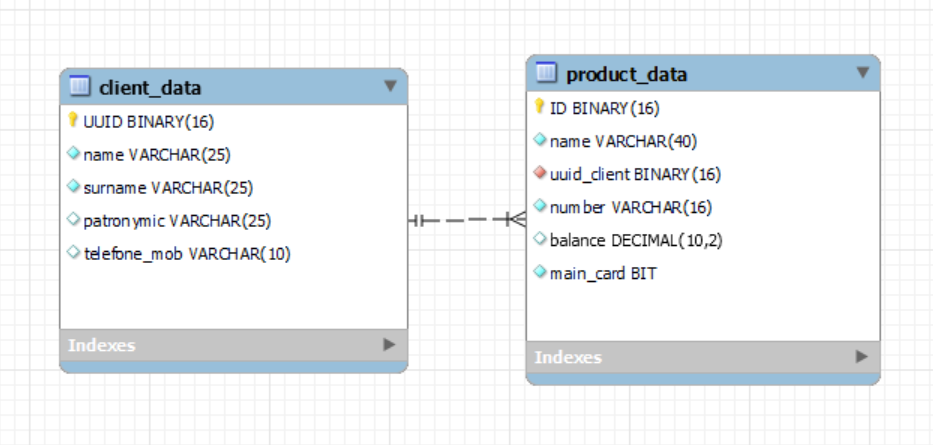
Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.

Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

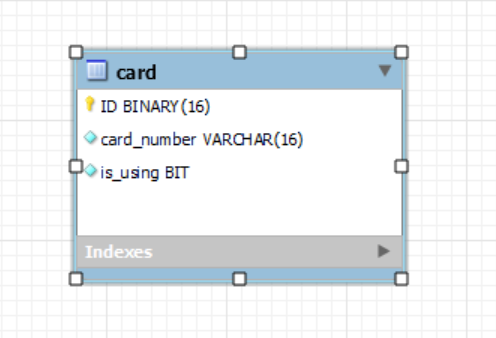
ER-диаграммы для основных сущностей;



*рис. 2 ERD clientWebService*

**

*рис. 3 ERD productProfileService*

**

*рис 4 ERD cardService*

Для генерации идентификатора выбран java.util.UUID для обеспечения децентрализованой генерации уникального ключа в распределенных системах.

Для хранения информации о балансе – денежных средствах выбран класс java.math.BigDecimal для хранения в базе данных decimal(10,2).

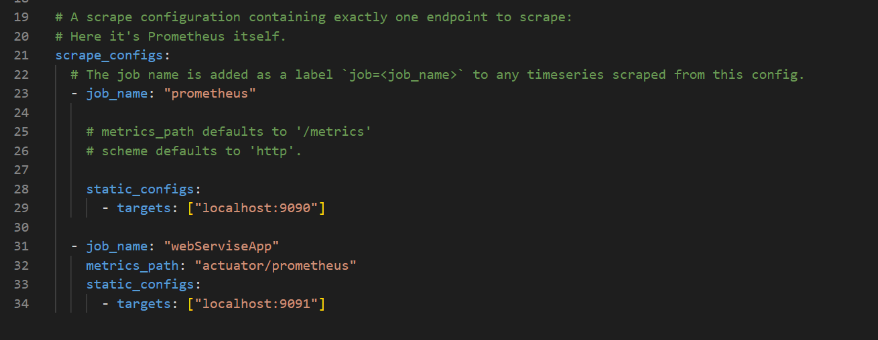
Для еще большей гибкости разработки добавлены интерфейсы прослойки между сервисами и контроллерами.

При копировании файла в локальное хранилище имя файла изменяется добавляя хэш в целях устранения конфликта имен.

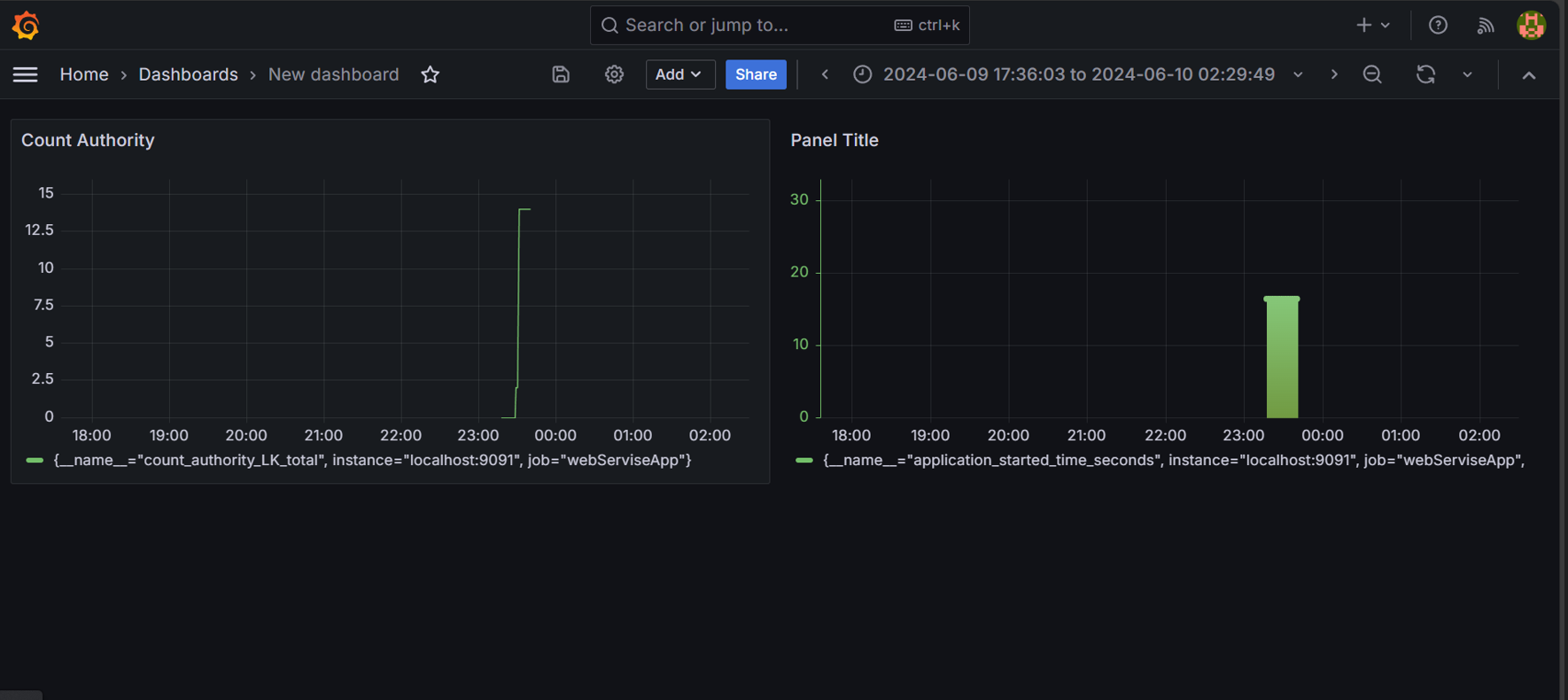
Для передачи импортируемого файла в сервис управления картами form-data используется org.springframework.web.multipart.MultipartFile.

Для валидации и обработки файла созданы кастомные ошибки и отдельный сервис для парсинга формата csv.

Для мониторинга систем к проекту подключены и настроены Prometheus и Grafana.



*рис 5 конфиг Prometheus*

**

*рис 6 Дашборд Grafana*

MVP версия размещена на GIT https://github.com/EvgeniyVadimovichTitov/diplomProject.git.

**ЗАКЛЮЧЕНИЕ**

В ходе проекта была проведена разработка микросервисов для создания банковского сервиса на Java Spring Boot. Проект включал в себя разработку сервисов управления продуктовым профилем клиента, управления банковскими картами, API Gateway, Eruika, веб-сервиса клиента и веб-сервиса администратора.

Основные моменты проектирования микросервисов:

Разделение системы на независимые сервисы. Это позволило обеспечить гибкость и масштабируемость системы, а также упростить её обслуживание и развитие.

Использование Java Spring Boot. Выбор этой технологии позволил создать гибкую и легко масштабируемую систему, которая может быть развёрнута как на локальных серверах, так и в облаке.

Применение принципов RESTful API. Это обеспечило простоту интеграции с другими системами и возможность использования различных клиентских приложений.

Разработка сервисов с учётом бизнес-требований. Это позволило создать систему, отвечающую потребностям бизнеса и обеспечивающую эффективное управление банковским сервисом.

Создание MVP (Minimum Viable Product). Это позволило быстро вывести продукт на рынок и получить обратную связь от пользователей, что помогло улучшить систему и сделать её более удобной и функциональной.

Обеспечение безопасности данных. В проекте были реализованы меры по защите данных клиентов, включая шифрование, аутентификацию и авторизацию.

Тестирование и отладка. Были проведены тесты на функциональность, производительность и безопасность, чтобы убедиться в надёжности системы.

Документация. Была создана документация по каждому сервису, включая описание функций, примеры использования и инструкции по настройке.

Интеграция с внешними системами. Были разработаны механизмы интеграции с платёжными системами, CRM и другими сервисами, необходимыми для работы банковского сервиса.

Проект успешно реализован и может служить основой для дальнейшего развития и улучшения банковского сервиса. Разработанные микросервисы обеспечивают гибкость, масштабируемость и безопасность системы, что делает её привлекательной для внедрения в банковские организации.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Spring Initializr [Электронный ресурс ] // Официальный сайт Spring. Режим доступа: https://spring.io/blog/2019/03/05/spring- initializr-new-ui (дата обращения: 16.04.2019).

2. Карл Вигерс и Джой Битти «Разработка требований к программному обеспечению» 3 издание дополненное 2024 г

3. Rest API Best Practices [Электронный ресурс ] // Сайт habrahabr. Режим доступа: https://habr.com/ru/post/351890/ /дата обращения: 17.04.2019)

4. Project Lombok [ Электронный ресурс] // Официальный сайт Lombok. Режим доступа: https://projectlombok.org/ /дат а обращения: 17.04.2019)

5. Maven [ Элек тронный ресурс] // Официальный сайт Maven. Режим доступа: https://maven.apache.org/ /дата обращения: 17.04.2019)

6. Gradle [ Электронный ресурс] // Официальный сайт Gradle. Режим доступа: https://gradle.org/ /дата об ращения: 20.04.2019)

7. Swagger [ Электронный ресурс] // Официальный сайт Swagger. Режим доступа: https://swagger.io/ /дата обращения 21.03.2019)

8. Rami [Электронный ресурс] // Официальный сайт RAML. Режим доступа: https://raml.org/ /дата обращения 2.05.2019)

9. Spring Boot от начала и до продакшена [Электронный ресурс] // Сайт habrahabr. Режим доступа: https://habr.com/ru/post/257223/ /дата обращения: 5.05.2019).

10. Java [Электронный ресурс] // Официальная документация Java от компании Oracle- Режим доступа: https://docs.oracle.eom/javase/8/does/ (дата обращения: 10.05.2019).

11. Spring [Электронный ресурс] // Официальный сайт Spring - Режим доступа: https://spring.io/ (дата обращения: 10.05.2019). 12. Hibernate [Электронный ресурс] // Официальный сайт Hibernate - Режим доступа: https://hibemate.org/ /дата обращения: 14.05.2019).

13. JSON [Электронный ресурс] // Официальная документация JSON - Режим доступа: https://www.json.org/ (дата обращения: 10.06.2019).

14. XML [Электронный ресурс] // Официальная документация XML - Режим доступа: https://www.w3.org/XML/ (дата обращения: 10.06.2019).

15. JavaPoet [Электронный ресурс] // Страница на GitHub - Режим доступа: https://github.com/square/javapoet /дата обращения: 10.06.2019).

16. TechRocks [Электронный ресурс] // Официальный сайт TechRocks - режим доступа https://techrocks.ru/2018/07/29/programming-languages-popularity-2018/ (дата обращения: 11.06.2019).

17. MDN [Электронный ресурс] // Официальный сайт MDN - режим доступа httpsV/developer.mozilla.org/m/docs/Web/HTTP/Methods (дата обращения: 11.06.2019) .

**ПРИЛОЖЕНИЯ**

Приложение №1

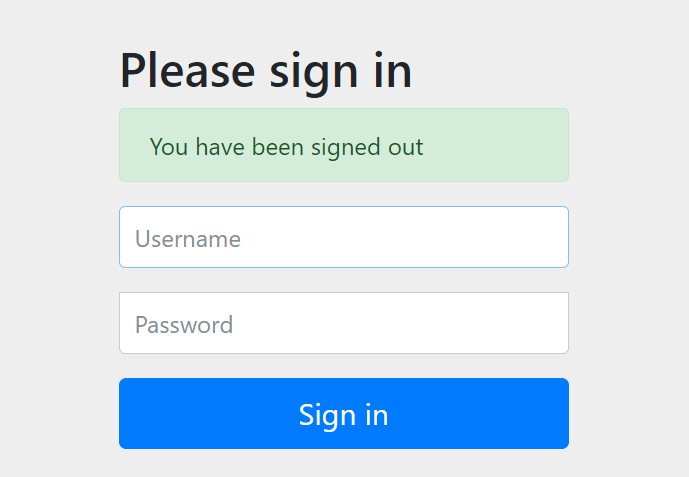
Спецификация требований к банковскому сервису

1. Введение

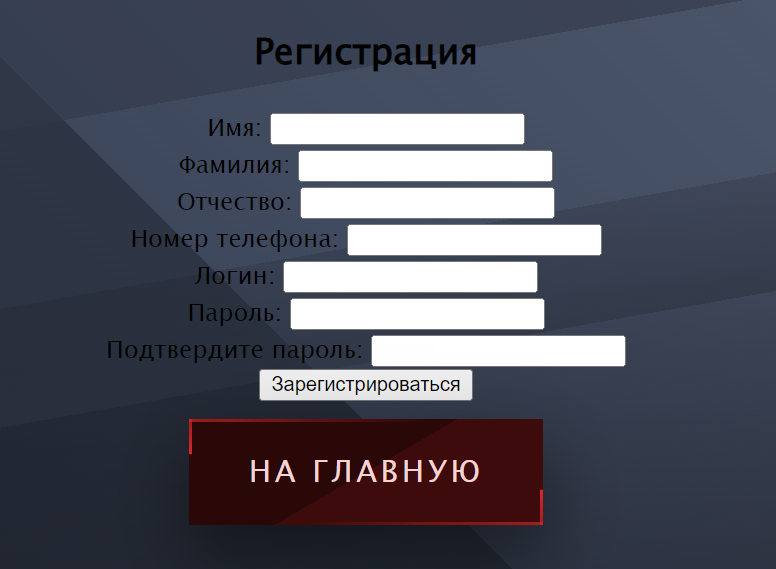
В данном документе представлены требования к разработке банковского сервиса, который будет предоставлять клиентам доступ к информации о счетах, вкладах и других продуктах банка, а также возможность проведения операций по счетам (пополнение, снятие, перевод средств). Сервис должен иметь удобный и понятный интерфейс, быть безопасным и надёжным.

2. Функциональные требования

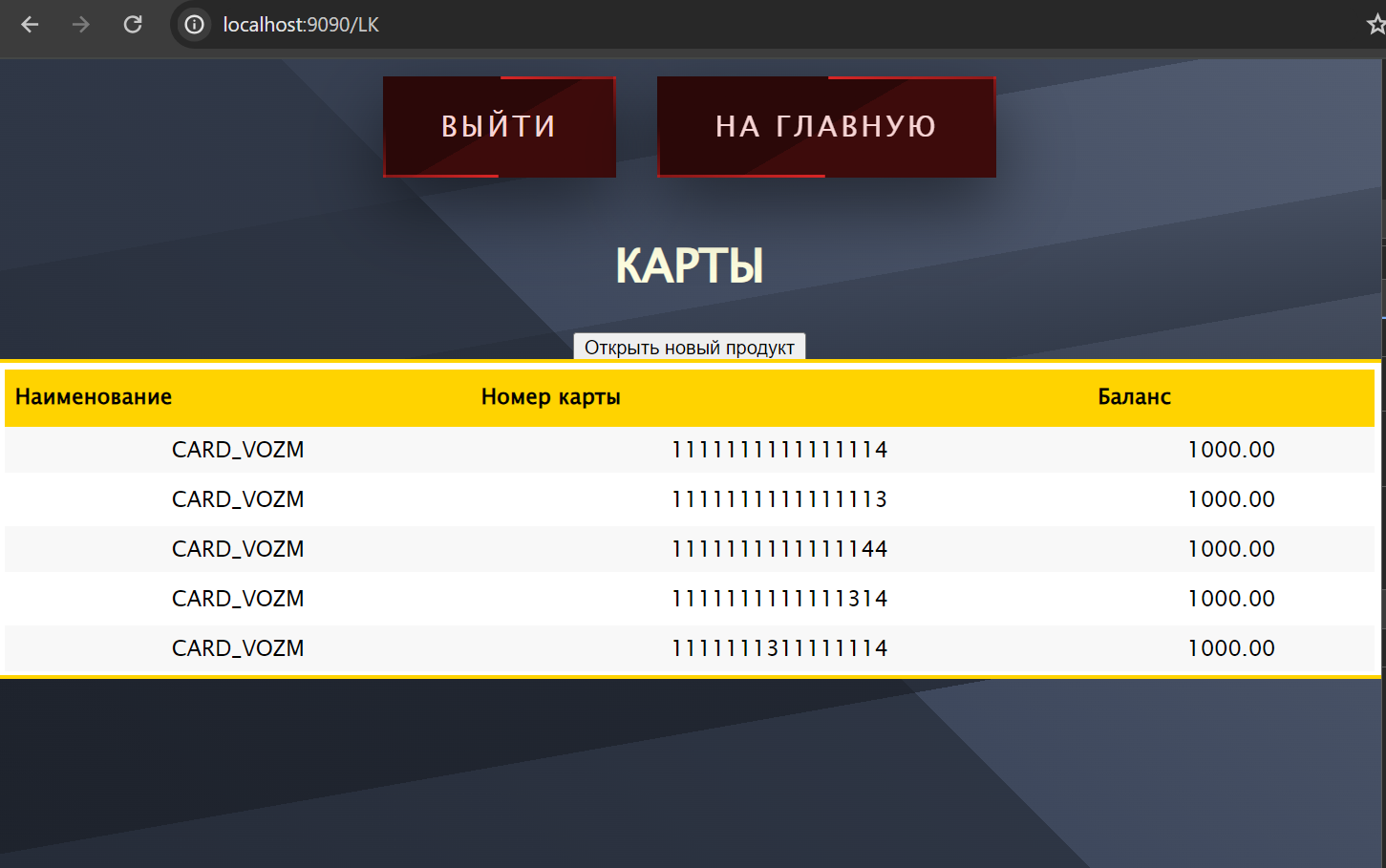
2.1. Регистрация и авторизация пользователей:



Возможность регистрации новых пользователей с использованием электронной почты или номера телефона;



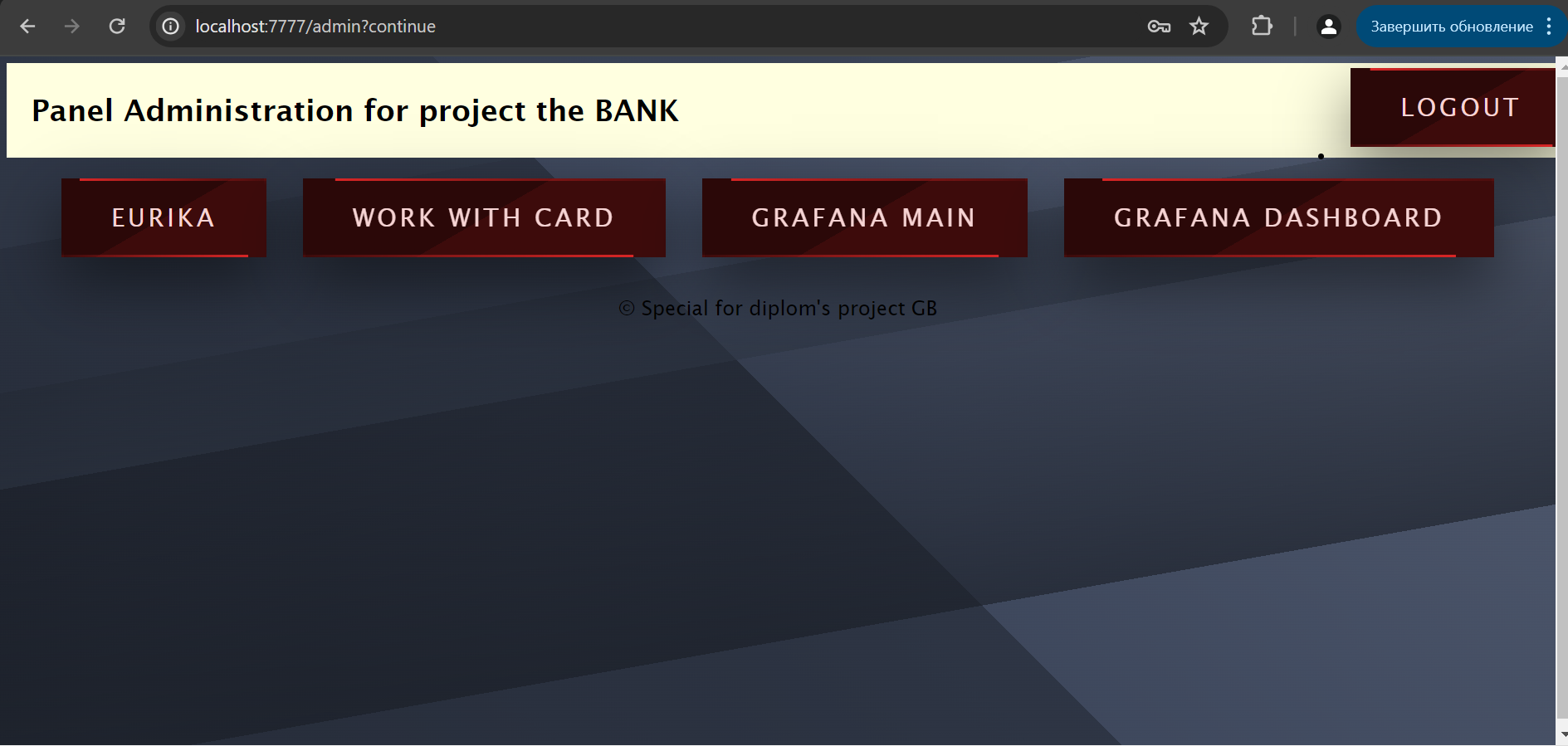
Интерфейс личного кабинета клиента



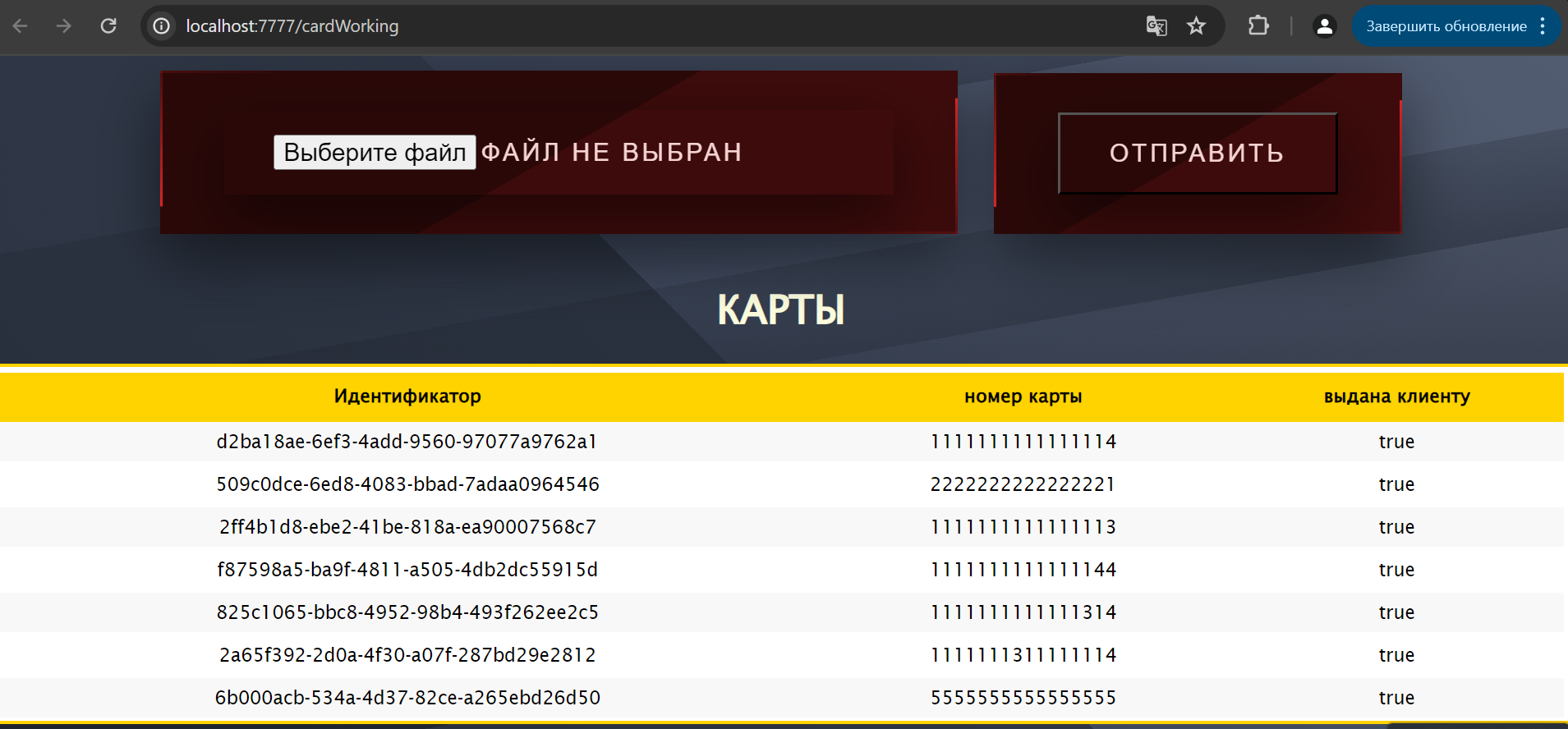
Поддержка различных способов аутентификации (пароль, двухфакторная аутентификация);

Автоматическое обновление пароля при его смене пользователем.

Интерфейс панели администартора.



Возможность импорта реестра карт в формате csv и мониторинг текущего состояния карт.



Копирование импортированного файла в локальное хранилище.

2.2. Просмотр информации о счёте:

Отображение баланса счёта, истории операций, процентных ставок и других параметров;

Возможность просмотра информации в режиме реального времени или за определённый период.

2.3. Проведение операций по счёту:

Пополнение счёта с помощью перевода средств с другого счёта или оплаты картой;

Снятие наличных в банкоматах или кассах банка;

Перевод средств между счетами внутри банка или на счета в других банках.

2.4. Управление платежами:

Настройка автоматических платежей по расписанию;

Отслеживание статуса платежей;

Отмена или изменение платежей.

2.5. Безопасность и конфиденциальность:

Шифрование данных при передаче по сети;

Защита от несанкционированного доступа к данным;

Соблюдение требований законодательства о защите персональных данных.

3. Нефункциональные требования

3.1. Производительность:

Время отклика на запросы не более 1 секунды;

Обработка большого количества запросов без потери производительности.

3.2. Масштабируемость:

Способность системы обрабатывать растущее количество пользователей и транзакций;

Простота добавления новых серверов для увеличения производительности.

3.3. Надёжность:

Отсутствие простоев системы из-за технических проблем;

Резервное копирование данных для восстановления после сбоев.

3.4. Удобство использования:

Простой и интуитивно понятный интерфейс;

Наличие помощи и поддержки для пользователей.

4. Требования к интерфейсу

4.1. Дизайн:

Современный и привлекательный дизайн;

Адаптивный дизайн для мобильных устройств.

4.2. Навигация:

Логичная и понятная структура

продолжи

4. Требования к интерфейсу

4.1. Дизайн:

Современный и привлекательный дизайн;

Адаптивный дизайн для мобильных устройств;

Соблюдение корпоративного стиля банка.

4.2. Навигация:

Логичная и понятная структура;

Возможность поиска по ключевым словам;

Интуитивно понятный интерфейс для выполнения операций.

5. Тестирование

5.1. Модульное тестирование:

Проверка отдельных компонентов системы на соответствие функциональным требованиям;

Использование тестовых данных для проверки корректности работы функций.

5.2. Интеграционное тестирование:

Объединение модулей в систему и проверка их взаимодействия;

Выявление ошибок, связанных с обменом данными между модулями.

5.3. Функциональное тестирование:

Выполнение реальных сценариев использования сервиса;

Проверка соответствия системы функциональным требованиям.

6. Документация

6.1. Руководство пользователя:

Описание основных функций сервиса;

Инструкции по регистрации, авторизации и использованию сервиса.

6.2. Техническая документация:

Спецификации API для разработчиков;

Архитектура системы и описание компонентов;

Конфигурационные файлы и параметры настройки.

7. Поддержка и обслуживание

7.1. Служба поддержки:

Предоставление помощи пользователям по телефону, электронной почте или через чат;

Обработка запросов на изменение или улучшение сервиса.

7.2. Обновление и исправление ошибок:

Регулярное обновление системы с учётом новых требований и изменений в законодательстве;

Исправление обнаруженных ошибок и проблем с производительностью.

Эти требования являются основой для разработки банковского сервиса, который будет соответствовать потребностям клиентов и обеспечивать безопасность и надёжность операций.

продолжи

8. Архитектура и технологии

8.1. Выбор стека технологий:

Java Spring Boot для разработки серверной части;

React или Angular для создания пользовательского интерфейса;

PostgreSQL для хранения данных;

Redis для кэширования данных.

8.2. Архитектура системы:

Микросервисная архитектура с разделением на сервисы по функциональности (например, сервис аутентификации, сервис операций по счёту и т. д.);

Взаимодействие между сервисами через REST API.

9. Безопасность

9.1. Шифрование данных:

Использование SSL/TLS для защиты данных при передаче по сети;

Шифрование конфиденциальных данных (например, паролей) при хранении в базе данных.

9.2. Защита от несанкционированного доступа:

Аутентификация и авторизация пользователей;

Ограничение доступа к данным в зависимости от роли пользователя.

9.3. Соблюдение законодательства:

Обработка персональных данных в соответствии с требованиями GDPR;

Защита от мошенничества и отмывания денег.

10. Мониторинг и логирование

10.1. Сбор метрик:

Количество запросов к каждому сервису;

Время выполнения запросов;

Ошибки и исключения.

10.2. Логирование событий:

Регистрация действий пользователей;

Запись ошибок и исключений.

Эти меры позволят отслеживать работу системы, выявлять проблемы и принимать меры по их устранению.

11. Тестирование безопасности

11.1. Пентестинг:

Проверка системы на уязвимости к атакам злоумышленников;

Оценка уровня защиты от атак типа SQL-инъекция, XSS и других.

11.2. Тестирование на соответствие стандартам:

PCI DSS для обработки платёжной информации;

ISO 27001 для обеспечения информационной безопасности.

Тестирование безопасности позволит убедиться в том, что система соответствует требованиям законодательства и обеспечивает защиту данных клиентов.

Приложение № 2

**Спецификация API сервиса управления данными о картах**

Название API: OpenAPI definition

Версия API: v1

Описание: API для управления данными о картах.

URL сервера: http://localhost:8082

Методы:

PUT /api/v1/card/setUsingCard: устанавливает статус использования карты. Требуется тело запроса в формате JSON с данными о карте. Ответ возвращает данные о карте в формате JSON.

POST /api/v1/card/importFileCsv: импортирует данные из CSV-файла. Требуется тело запроса с файлом в двоичном формате. Ответ возвращает объект с описанием результата импорта.

GET /api/v1/card/getFreeCard: получает данные о свободной карте. Ответ возвращает данные о карте в формате JSON.

GET /api/v1/card/getCardByNumber/{cardNumber}: получает данные о карте по номеру. Ответ возвращает данные о карте в формате JSON.

GET /api/v1/card/getAll: получает все данные о картах. Ответ возвращает массив данных о картах в формате JSON.

Типы данных:

CardData: объект с данными о карте, включая идентификатор, номер карты и статус использования.

ResponseEntityResultImport: объект с результатом импорта данных из CSV-файла, включая описание и код ошибки.

Swagger-schema:

{

"openapi": "3.0.1",

"info": {

"title": "OpenAPI definition",

"version": "v1"

},

"servers": [

{

"url": "http://localhost:8082",

"description": "Generated server url"

}

],

"paths": {

"/api/v1/card/setUsingCard": {

"put": {

"tags": [

"controller-card-date"

],

"operationId": "setIsingCard",

"requestBody": {

"content": {

"application/json": {

"schema": {

"$ref": "#/components/schemas/CardData"

}

}

},

"required": true

},

"responses": {

"200": {

"description": "OK",

"content": {

"\*/\*": {

"schema": {

"$ref": "#/components/schemas/CardData"

}

}

}

}

}

}

},

"/api/v1/card/importFileCsv": {

"post": {

"tags": [

"controller-card-date"

],

"operationId": "importFileCsv",

"requestBody": {

"content": {

"application/json": {

"schema": {

"required": [

"file"

],

"type": "object",

"properties": {

"file": {

"type": "string",

"format": "binary"

}

}

}

}

}

},

"responses": {

"200": {

"description": "OK",

"content": {

"\*/\*": {

"schema": {

"$ref": "#/components/schemas/ResponseEntityResultImport"

}

}

}

}

}

}

},

"/api/v1/card/getFreeCard": {

"get": {

"tags": [

"controller-card-date"

],

"operationId": "getFreeCard",

"responses": {

"200": {

"description": "OK",

"content": {

"\*/\*": {

"schema": {

"$ref": "#/components/schemas/CardData"

}

}

}

}

}

}

},

"/api/v1/card/getCardByNumber/{cardNumber}": {

"get": {

"tags": [

"controller-card-date"

],

"operationId": "getCardByNumber",

"parameters": [],

"responses": {

"200": {

"description": "OK",

"content": {

"\*/\*": {

"schema": {

"$ref": "#/components/schemas/CardData"

}

}

}

}

}

}

},

"/api/v1/card/getAll": {

"get": {

"tags": [

"controller-card-date"

],

"operationId": "getAll",

"responses": {

"200": {

"description": "OK",

"content": {

"\*/\*": {

"schema": {

"type": "array",

"items": {

"$ref": "#/components/schemas/CardData"

}

}

}

}

}

}

}

}

},

"components": {

"schemas": {

"CardData": {

"type": "object",

"properties": {

"id": {

"type": "string",

"format": "uuid"

},

"cardNumber": {

"type": "string"

},

"isUsing": {

"type": "boolean"

}

}

},

"ResponseEntityResultImport": {

"type": "object",

"properties": {

"description": {

"type": "string"

},

"errCode": {

"type": "integer",

"format": "int32"

}

}

}

}

}

}

Приложение № 3

**Спецификация API Сервиса управления   
продуктовым профилем клиента**

Название API: OpenAPI definition

Версия API: v1

Описание API:

API предоставляет методы для работы с данными профиля и продуктов.

Методы позволяют создавать, обновлять и получать данные профиля и продуктов.

Доступные методы:

PUT /api/v1/profile/updateProfile: метод позволяет обновить данные профиля.

POST /api/v1/profile/product/addProduct: метод добавляет новый продукт в профиль клиента.

POST /api/v1/profile/createProfile: метод создаёт новый профиль.

GET /api/v1/profile/getAll: метод возвращает список всех продуктов.

GET /api/v1/profile/getProfileByUuid/{uuid}: метод получает данные профиля по UUID.

GET /api/v1/profile/getProfileByPhoneNumber/{phoneNumber}: метод получает данные профиля по номеру телефона.

Формат данных:

Данные передаются в формате JSON.

Типы данных:

В API используются следующие типы данных:

string: строка.

boolean: логическое значение.

number: число.

array: массив.

object: объект.

Сервер:

Сервер доступен по адресу <http://localhost:8081>.

Swagger-schema:

{

"openapi": "3.0.1",

"info": {

"title": "OpenAPI definition",

"version": "v1"

},

"servers": [

{

"url": "http://localhost:8081",

"description": "Generated server url"

}

],

"paths": {

"/api/v1/profile/updateProfile": {

"put": {

"tags": [

"rest-client-profile-controller"

],

"operationId": "updateClientData",

"requestBody": {

"content": {

"application/json": {

"schema": {

"$ref": "#/components/schemas/ClientData"

}

}

},

"required": true

},

"responses": {

"200": {

"description": "OK",

"content": {

"\*/\*": {

"schema": {

"$ref": "#/components/schemas/ClientData"

}

}

}

}

}

}

},

"/api/v1/profile/product/addProduct": {

"post": {

"tags": [

"rest-product-data-controller"

],

"operationId": "addProduct",

"requestBody": {

"content": {

"application/json": {

"schema": {

"$ref": "#/components/schemas/ProductData"

}

}

},

"required": true

},

"responses": {

"200": {

"description": "OK",

"content": {

"\*/\*": {

"schema": {

"$ref": "#/components/schemas/ProductData"

}

}

}

}

}

}

},

"/api/v1/profile/createProfile": {

"post": {

"tags": [

"rest-client-profile-controller"

],

"operationId": "createClientData",

"requestBody": {

"content": {

"application/json": {

"schema": {

"$ref": "#/components/schemas/ClientData"

}

}

},

"required": true

},

"responses": {

"200": {

"description": "OK",

"content": {

"\*/\*": {

"schema": {

"$ref": "#/components/schemas/ClientData"

}

}

}

}

}

}

},

"/api/v1/profile/product/getAll": {

"get": {

"tags": [

"rest-product-data-controller"

],

"operationId": "getAll",

"responses": {

"200": {

"description": "OK",

"content": {

"\*/\*": {

"schema": {

"type": "array",

"items": {

"$ref": "#/components/schemas/ProductData"

}

}

}

}

}

}

}

},

"/api/v1/profile/getProfileByUuid/{uuid}": {

"get": {

"tags": [

"rest-client-profile-controller"

],

"operationId": "findById",

"parameters": [

{

"name": "uuid",

"in": "path",

"required": true,

"schema": {

"type": "string",

"format": "uuid"

}

}

],

"responses": {

"200": {

"description": "OK",

"content": {

"\*/\*": {

"schema": {

"$ref": "#/components/schemas/ClientData"

}

}

}

}

}

}

},

"/api/v1/profile/getProfileByPhoneNumber/{phoneNumber}": {

"get": {

"tags": [

"rest-client-profile-controller"

],

"operationId": "findByPhoneNumber",

"parameters": [

{

"name": "phoneNumber",

"in": "path",

"required": true,

"schema": {

"type": "string"

}

}

],

"responses": {

"200": {

"description": "OK",

"content": {

"\*/\*": {

"schema": {

"$ref": "#/components/schemas/ClientData"

}

}

}

}

}

}

}

},

"components": {

"schemas": {

"ClientData": {

"type": "object",

"properties": {

"uuid": {

"type": "string",

"format": "uuid"

},

"name": {

"type": "string"

},

"surname": {

"type": "string"

},

"patronymic": {

"type": "string"

},

"phoneNumber": {

"type": "string"

},

"products": {

"type": "array",

"items": {

"$ref": "#/components/schemas/Product"

}

}

}

},

"Product": {

"type": "object",

"properties": {

"id": {

"type": "string",

"format": "uuid"

},

"name": {

"type": "string",

"enum": [

"CARD\_PRIV",

"CARD\_VOZM"

]

},

"number": {

"type": "string"

},

"balance": {

"type": "number"

},

"mainCard": {

"type": "boolean"

}

}

},

"ProductData": {

"type": "object",

"properties": {

"id": {

"type": "string",

"format": "uuid"

},

"name": {

"type": "string",

"enum": [

"CARD\_PRIV",

"CARD\_VOZM"

]

},

"client": {

"type": "string",

"format": "uuid"

},

"number": {

"type": "string"

},

"balance": {

"type": "number"

},

"mainCard": {

"type": "boolean"

}

}

}

}

}

}

Приложение № 4

**Техническое задание на разработку банковского сервиса**

1. Введение

1.1. Наименование проекта: «Разработка банковского сервиса».

1.2. Заказчик: [Название компании].

1.3. Исполнитель: [Название исполнителя].

1.4. Сроки выполнения: [Начало и окончание работ].

2. Цели и задачи проекта

2.1. Цель проекта: разработка и внедрение банковского сервиса для управления продуктовым профилем клиента, банковскими картами, API Gateway, Eruika, веб-сервисом клиента и администратора.

2.2. Задачи проекта:

Разработка сервиса управления продуктовым профилем клиента.

Разработка сервиса управления банковскими картами.

Создание API Gateway для обеспечения взаимодействия между сервисами.

Реализация сервиса Eruika.

Проектирование и разработка веб-сервиса клиента.

Проектирование и разработка веб-сервиса администратора.

3. Требования к проекту

3.1. Технологии и инструменты:

Java, Spring Boot, Maven.

Thymeleaf для создания шаблонов HTML.

Spring Security для обеспечения безопасности.

Spring Data JPA для работы с базой данных.

Lombok для упрощения кода.

Spring Cloud для обеспечения масштабируемости и отказоустойчивости.

Prometheus и Grafana для мониторинга и анализа производительности.

Spring Cloud Gateway для маршрутизации запросов.

3.2. Требования

3.2.1. Функциональные требования:

Регистрация и авторизация пользователей;

Управление продуктовым профилем клиента (выбор продуктов, изменение настроек);

Управление банковскими картами (выпуск, блокировка, смена PIN-кода);

Получение информации о продуктах и услугах банка;

Оплата товаров и услуг через банковские карты;

Просмотр истории операций по картам;

Настройка уведомлений о событиях по картам.

3.2.2. Нефункциональные требования:

Безопасность данных и операций;

Масштабируемость и отказоустойчивость;

Поддержка различных устройств и браузеров;

Интеграция с другими системами банка (CRM, биллинг и т.д.);

Возможность мониторинга и анализа данных.

3.3. Архитектура системы:

Сервис управления продуктовым профилем должен быть реализован как RESTful API.

Сервис управления банковскими картами должен обеспечивать возможность создания, обновления и удаления карт.

API Gateway должен обеспечивать маршрутизацию запросов между сервисами.

Eruika должен предоставлять информацию о продуктах и услугах банка.

Веб-сервис клиента должен обеспечивать доступ к информации о продуктах и услугах, а также возможность оформления заявок на услуги.

Веб-сервис администратора должен обеспечивать управление пользователями, ролями и разрешениями.

3.4. Функциональность:

Регистрация и аутентификация пользователей.

Управление продуктовым профилем.

Оформление заявок на банковские продукты и услуги.

Просмотр истории операций по картам.

Получение информации о доступных продуктах и услугах.

Мониторинг и анализ производительности сервисов.

4. Этапы разработки

4.1. Анализ требований и проектирование архитектуры;

4.2. Разработка сервисов и API;

4.3. Тестирование и отладка;

4.4. Развертывание и интеграция с другими системами;

4.5. Мониторинг и анализ данных.

5. Приёмка:

Проведение тестирования и верификации разработанного сервиса.

Оценка соответствия сервиса требованиям ТЗ и ожиданиям заказчика.

Подписание акта приёмки-передачи между заказчиком и исполнителем.

Внесение изменений:

В случае обнаружения несоответствий или ошибок после приёмки, внесение необходимых изменений в сервис.

5.1.Финальная проверка:

Окончательная проверка сервиса перед его выпуском в эксплуатацию.

Обеспечение полной готовности сервиса к работе.

5.2. Релиз:

Запуск сервиса в работу и предоставление доступа пользователям.

Мониторинг работы сервиса и сбор обратной связи от пользователей.

1. Mission Critical System Margaret Rouse - https://www.techopedia.com/definition/23583/mission-critical-system [↑](#footnote-ref-1)
2. Просто о микросервисах - https://habr.com/ru/companies/raiffeisenbank/articles/346380/ [↑](#footnote-ref-2)
3. Карл Вигерс и Джой Битти «Разработка требований к программному обеспечению» 3 издание дополненное 2024 г – стр. 6. [↑](#footnote-ref-3)
4. Garafana - https://ru.wikipedia.org/wiki/Grafana [↑](#footnote-ref-4)
5. MVC - https://ru.wikipedia.org/wiki/Model-View-Controller [↑](#footnote-ref-5)