Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 24.М71-мм

Реализация OAuth 2.0 в основной системе электронных денег в соответствии со стандартами BI SNAP.

Юсуп Амин Турмуди

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
доцент кафедры системного программирования, к. Ф.-м. н.доцент Д. В. Луцив

Санкт-Петербург
2025

Оглавление

Введение

Развитие интернета — одно из самых значительных достижений человечества в 20 веке [8]. С помощью интернета люди могут мгновенно получать доступ к информации, знаниям, услугам и даже товарам из различных уголков мира. Это удобство доступа стало движущей силой крупных изменений в бизнесе — перехода от традиционных (оффлайн) бизнес-моделей к цифровым (онлайн) бизнес-моделям, с основной целью охвата более широких рынков и достижения операционной эффективности.

Однако расширение рынка достигается не только через онлайн-сервисы, ориентированные непосредственно на конечных потребителей, но и через сотрудничество с другими компаниями в рамках схемы B2B[1]. В экосистеме B2B компании работают вместе, чтобы предоставлять услуги или обмениваться данными с целью укрепления своих бизнес-позиций. Это делает интеграцию систем приложений между компаниями необходимостью, которую невозможно избежать.

С увеличением сложности взаимодействий между компаниями вопрос безопасности становится первостепенным. В отличие от внутренних систем, которые включают только стороны внутри компании, системы B2B позволяют внешним сторонам получать доступ и даже управлять определенными данными. Поэтому системы B2B должны внедрять механизмы аутентификации (для проверки того, кто имеет доступ) и авторизации (для контроля того, что можно и нельзя использовать).

В этом контексте недостаточно просто знать, кто получает доступ к системе. Система также должна понимать, какие действия разрешено выполнять этой стороне на ресурсном сервере. Цель — обеспечить, чтобы партнеры B2B могли получать доступ к данным только в рамках предоставленных прав доступа, не затрагивая чувствительную или нерелевантную информацию. Некоторые широко используемые протоколы для защиты коммуникаций и доступа между приложениями вклю-

[1]Business to Business

чают OAuth 2.0 и SAML[2] 2.0.

SAML 2.0 часто используется в традиционных корпоративных приложениях, таких как системы управления персоналом, бэк-офисные банковские системы и тому подобное [6]. В то время как OAuth 2.0 более часто используется для защиты HTTP-сервисов, особенно API[3], которые являются основой обмена данными в современных приложениях.

В качестве примера, основная система приложения для электронных денег обычно реализует традиционные стандарты безопасности. В этой архитектуре каждый клиент, запрашивающий данные, включает API-ключ, который затем проверяется ресурсным сервером. Если API-ключ действителен, запрашиваемые данные обрабатываются и возвращаются клиенту. Хотя этот подход может быть еще допустим в рамках внутренней системы компании, риски безопасности увеличиваются, когда система электронных денег начинает интегрироваться с внешними сторонами в рамках B2B.

Таким образом, необходимо обновление системы безопасности, чтобы гарантировать защиту ресурсного сервера, даже если он открыт для сотрудничества с другими партнерами. Одним из решений, которое можно внедрить, является использование протокола OAuth 2.0.

В этой статье будет рассмотрено внедрение концепций аутентификации и авторизации с использованием протокола OAuth 2.0. Реализация будет проводиться с использованием языка программирования Java с использованием фреймворка Javax Servlet. Этот проект предназначен для защиты ресурсного сервера в сценарии B2B-приложений, чтобы доступ к данным мог осуществляться безопасно, контролируемо и в соответствии с современными отраслевыми стандартами.

[2]Security Assertion Markup Language

[3]Application Programming Interface

1. Постановка задачи

Основная цель данной работы — реализовать протокол OAuth 2.0 для обеспечения защиты ресурсного сервера в контексте приложения типа B2B. С помощью внедрения OAuth 2.0 система должна обеспечивать более безопасное и гибкое управление аутентификацией и авторизацией в соответствии с отраслевыми стандартами. Для достижения этой цели были определены следующие ключевые задачи:

- Разработать Authorization Server на основе OAuth 2.0 с использованием Javax Servlet для генерации безопасных токенов доступа, которые могут управляться на основе scope (областей доступа) и ролей пользователей.
- Интегрировать OAuth 2.0 в клиентское приложение и ресурсный сервер, а также провести end-to-end тестирование для обеспечения соответствия процессов аутентификации и авторизации стандарту протокола OAuth 2.0.
- Провести валидацию безопасности и эффективности реализации, включая обработку токенов, контроль доступа на основе scope и управление авторизацией для сторонних приложений.

2. Обзор

2.1. Традиционные методы аутентификации API

Взаимодействие между API внутри компании играет важную роль в интеграции систем и обмене данными между различными отделами. Основная цель такой коммуникации — формирование более полной и полезной информации, которая может поддерживать процессы принятия решений. Например, интеграция между бухгалтерией и отделом кадров позволяет компании автоматически и точно отображать информацию о зарплате сотрудника, его стаже и занимаемой должности.

В контексте внутренних систем компании аспект безопасности при обмене API обычно не отличается высокой сложностью. Поскольку все системы функционируют в контролируемой среде, традиционные методы аутентификации считаются достаточными. Одним из наиболее часто используемых методов аутентификации в таких случаях является API-ключ.

API-ключ — это уникальный токен, который клиент использует для доступа к API [1]. Обычно этот токен представляет собой случайную строку или определённый формат кодирования, например Base64. Механизм его работы достаточно прост: когда клиент хочет получить доступ к ресурсу сервера, он должен включить API-ключ в каждый запрос. Сервер затем проверяет API-ключ, чтобы убедиться, что запрос поступил от авторизованного источника. Ниже представлена базовая иллюстрация того, как работает API-ключ:

- Клиент отправляет запрос на определённую конечную точку, включая API-ключ в заголовок запроса или как параметр URL[4].
- Ресурсный сервер проверяет действительность API-ключа.
- Если API-ключ действителен — предоставляется доступ к ресурсу; если нет — запрос отклоняется.

[4]Uniform Resource Locator

Хотя этот метод достаточно эффективен в ограниченных сценариях, таких как внутренняя среда компании, у него есть несколько существенных недостатков. API-ключи являются статичными и не распознают личность пользователя или уровень доступа. Кроме того, если API-ключ попадёт в руки злоумышленников, неавторизованные лица смогут легко получить доступ ко всем ресурсам без каких-либо ограничений.

2.2. JWT

JWT[5]— это открытый стандарт, определённый в RFC 7519, который предоставляет компактный и автономный механизм для безопасной передачи информации между сторонами в формате JSON-объекта [9]. JWT разработан таким образом, чтобы содержащаяся в нём информация могла быть проверена и заслуживала доверия, поскольку она подписана с использованием криптографических методов. Подпись может быть создана с помощью секретного ключа с алгоритмом HMAC[6] или пары открытого и закрытого ключей с использованием алгоритмов RSA[7] или ECDSA[8].

В последние годы использование JWT стало всё более популярным при разработке современных приложений, особенно веб- и мобильных. Такая популярность объясняется, в том числе, простотой внедрения и высоким уровнем безопасности при передаче данных между компонентами системы. Кроме того, JWT не требует хранения состояния , что делает его особенно подходящим для использования в мобильных приложениях, распределённых архитектурах, таких как микросервисы, и одностраничных приложениях.

Одним из практических примеров использования JWT является мобильное банковское приложение. Применение JWT в этом контексте основано на схожих принципах с его использованием в интернет-банкинге

[5]JSON Web Token

[6]Hash-based Message Authentication Code

[7]Rivest–Shamir–Adleman

[8]Elliptic Curve Digital Signature Algorithm

через веб-браузер, однако из-за различных характеристик и архитектуры мобильных приложений требуются некоторые важные адаптации.

Когда пользователь открывает мобильное банковское приложение и успешно входит в систему, используя действительные учетные данные — такие как имя пользователя и пароль или биометрическую аутентификацию отпечаток пальца или распознавание лица, — сервер генерирует JWT, содержащий информацию об аутентификации пользователя. Этот токен включает идентификатор пользователя, его роли или права доступа, а также срок действия токена. После успешного создания токен отправляется обратно в мобильное приложение в составе ответа сервера.

В отличие от браузеров, которые обычно хранят токены в cookies, мобильные приложения, как правило, хранят JWT в защищённом локальном хранилище, таком как Secure Storage или Keychain для iOS, а также EncryptedSharedPreferences или Android Keystore для Android. Эти механизмы хранения разработаны для предотвращения несанкционированного доступа, в том числе со стороны других приложений или при попытках эксплуатации на устройствах с root-доступом или jailbreak'ом.

Каждый раз, когда мобильное приложение отправляет запрос на сервер — например, чтобы проверить баланс счёта, просмотреть историю транзакций или выполнить перевод средств, — сохранённый JWT прикрепляется к заголовку Authorization с использованием схемы Bearer. Сервер получает запрос, проверяет подлинность токена с помощью зашифрованной подписи и удостоверяется, что срок действия токена ещё не истёк.

Если токен действителен и не просрочен, сервер обрабатывает запрос и возвращает соответствующий ответ. Однако если токен недействителен — например, если он был подделан, повреждён или срок его действия истёк, — сервер отклонит запрос и потребует от приложения повторной аутентификации пользователя.

Несмотря на то что JWT весьма эффективно используется для аутентификации конечных пользователей , в более масштабных сце-

нариях, таких как интеграция систем B2B, применение OAuth 2.0 в качестве рамочной системы авторизации значительно важнее и настоятельно рекомендуется.

3. Описание решения

Для защиты и авторизации доступа к серверу ресурсов во внутренней среде — всё ещё в рамках одной компании — использование API-ключей и JSON Web Tokens (JWT) обычно является достаточным. Оба механизма обеспечивают базовый контроль над тем, кто может получить доступ к API, и позволяют эффективно проверять личность клиента. JWT, в частности, предоставляет преимущества за счёт возможности включать утверждения (claims) и задавать срок действия токена, что делает его удобным для множества внутренних задач авторизации.

Однако, когда система развивается и начинает взаимодействовать с внешними системами — будь то между отделами, дочерними компаниями или даже сторонними организациями — одних только API-ключей и JWT уже недостаточно. Основная причина в трудностях управления авторизацией, отзывом токенов, более детализированным контролем доступа и необходимости получения явного согласия от владельца ресурса.

Именно здесь OAuth 2.0 становится более комплексным решением. OAuth 2.0 — это фреймворк авторизации, который позволяет безопасно получать доступ к ресурсам без прямой передачи учётных данных пользователя [2]. С помощью OAuth владелец ресурса (обычно пользователь) может предоставить ограниченные права доступа клиенту (стороннему приложению) через механизм токенов авторизации. Это особенно полезно в системах с участием нескольких организаций, где требуется гибкое и стандартизированное управление доступом.

Кроме того, OAuth 2.0 поддерживает различные потоки авторизации, такие как: Authorization Code Flow, Client Credentials Flow, Implicit Flow. Эти потоки можно адаптировать под конкретные сценарии — будь то веб-приложения, мобильные приложения или взаимодействие сервер-сервер. Всё это делает OAuth 2.0 рекомендуемым стандартом для интеграции между системами, где требуется высокий уровень безопасности, детализированная авторизация и масштабируемость в долгосрочной перспективе.

В заключение, для простых задач внутренней интеграции API-ключи и JWT остаются актуальными и эффективными. Однако для более сложных сценариев авторизации с участием нескольких сторон внедрение OAuth 2.0 — это стратегический шаг, соответствующий современным лучшим практикам в области безопасности.

4. Реализация

4.1. Архитектура приложения

С архитектурной точки зрения сервер авторизации реализован с использованием монолитной архитектуры. Такой подход был выбран, поскольку он обеспечивает простоту в развертывании, мониторинге и общем управлении системой. На ранних этапах разработки — особенно для стартапов или развивающихся компаний — операционная простота является приоритетом, позволяя команде сосредоточиться на проверке продукта и разработке основных функций.

Помимо удобства развертывания, выбор монолитной архитектуры также был обусловлен ограниченными человеческими ресурсами. При наличии относительно небольшой команды разработчиков создание системы авторизации как единого приложения позволяет централизовать и упростить процесс разработки. Кроме того, это облегчает коммуникацию между разработчиками, поскольку отпадает необходимость в управлении интеграцией множества отдельных сервисов, как это было бы в случае с микросервисной архитектурой.

Монолитный подход также предоставляет преимущества в области отладки и устранения неполадок. Когда возникают ошибки в процессе аутентификации или авторизации, поиск первопричины происходит быстрее, так как все логи и функции находятся в пределах одного приложения. Это особенно важно на раннем этапе разработки, когда стабильность системы ещё не достигнута и требуется оперативное внесение исправлений.

Тем не менее, даже монолитная архитектура разработана с учетом внутренней модульности, что позволяет при необходимости в будущем легко разделить её на отдельные сервисы по мере масштабирования приложения и роста потребностей бизнеса. Иными словами, использование монолита — это не жёсткое долгосрочное решение, а реалистичная начальная стратегия, сохраняющая гибкость для будущего развития системы и организации.

4.2. Диаграмма вариантов использования сервера OAuth

После определения архитектуры приложения следующим шагом является понимание общей схемы работы приложения, которое будет интегрировано с e-money через OAuth. Этот процесс необходим для того, чтобы каждый компонент приложения понимал свою роль — будь то аутентификация, авторизация или взаимодействие между системами. При наличии четкой схемы можно минимизировать вероятность ошибок или конфликтов между сервисами ещё на этапе проектирования.

На начальном этапе приложение будет взаимодействовать с сервером авторизации для получения B2B-токена в качестве исходной идентификации. После этого, для выполнения транзакций, более специфичных для конкретного пользователя, приложение запросит B2B2C-токен, который напрямую связан с аккаунтом клиента. Этот токен затем используется для выполнения различных операций, таких как проверка баланса или перевод средств. Такой процесс гарантирует, что каждый запрос, отправляемый на сторонний e-money сервер, проходит проверку и соответствует предоставленным правам доступа. Чтобы прояснить вышеизложенное, ниже представлена диаграмма вариантов использования, иллюстрирующая поток системы.

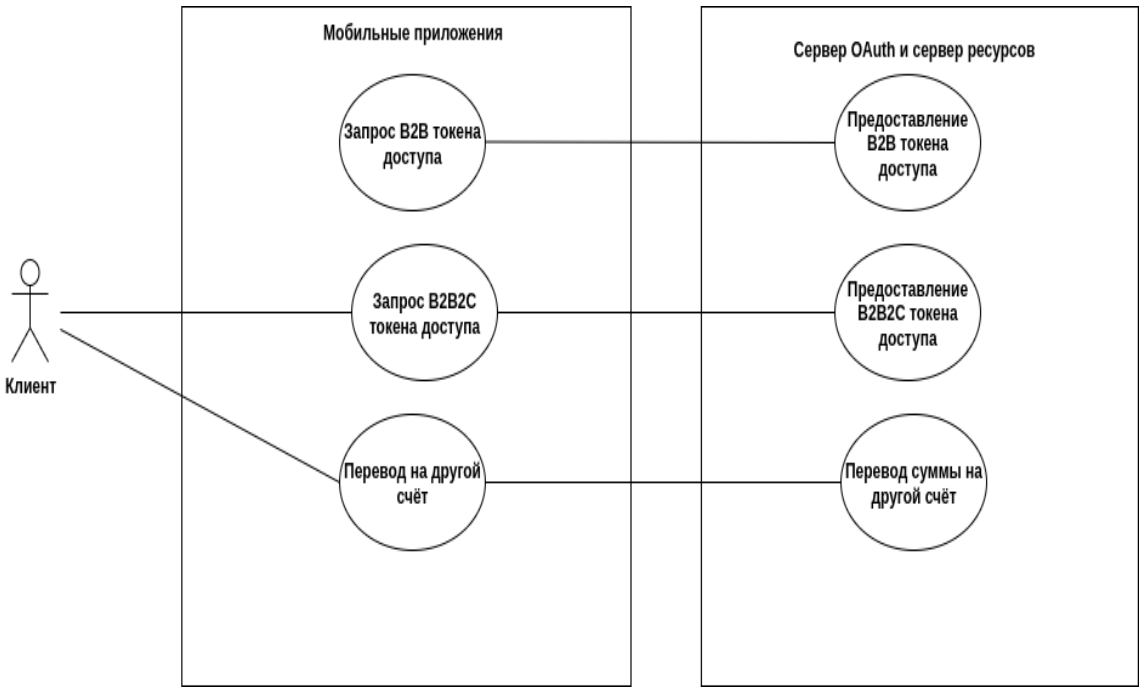

Рис. 1: Диаграмма вариантов использования сервера OAuth

4.3. Библиотека, используемая в проекте

Существует множество языков программирования, которые можно использовать для реализации упомянутых ранее трёх механизмов безопасности, включая Java, Python, Go, PHP, Ruby и другие. Хотя все эти языки реализуют одни и те же базовые концепции безопасности, они различаются по синтаксису, структуре и поддержке со стороны сообщества. При выборе языка для реализации следует учитывать несколько факторов Простота реализации, Наличие встроенных механизмов безопасности, Удобство обновления зависимостей, Простота поддержки и сопровождения Размер и активность сообщества, что помогает быстрее решать возникающие проблемы.В данном случае я выбрал язык программирования Java, используя javax.servlet для обработки HTTP-запросов и Jetty Client для выполнения API-вызовов.

javax.servlet входит в спецификацию Java EE (в настоящее время Jakarta EE) и предоставляет мощный API для создания серверных веб-приложений на Java. Ключевые компоненты :

- HttpServlet – базовый класс для создания сервлетов, работающих

по протоколу HTTP.

- doGet() и doPost() – методы для обработки HTTP-запросов типов GET и POST.
- HttpServletRequest – обеспечивает доступ к данным входящего запроса (заголовки, параметры, тело запроса и т.д.).
- HttpServletResponse – позволяет отправлять данные обратно клиенту в виде HTTP-ответа.

Помимо базовой обработки запросов, javax.servlet также поддерживает защиту API, например, с помощью фильтрации запросов и контроля доступа — обеспечивая, что только авторизованные клиенты могут обращаться к защищённым ресурсам [7].

Jetty Client — это HTTP-клиентская библиотека, разработанная в рамках проекта Jetty (известного также своим лёгким сервером приложений на Java) [5]. В отличие от сервлетов, обрабатывающих входящие HTTP-запросы, Jetty Client используется для исходящих HTTP-запросов к другим сервисам. Его можно рассматривать как программный аналог таких инструментов, как Postman, но предназначенный для интеграции в рабочие приложения.Типичные сценарии использования Jetty Client:

- Отправка HTTP-запросов к внешним API.
- Организация межсервисного взаимодействия в архитектурах микросервисов.
- Поддержка безопасных, асинхронных или параллельных API-вызовов.

Комбинируя javax.servlet и Jetty Client, можно создавать надёжные и безопасные приложения, которые как принимают внешние HTTP-запросы, так и инициируют их, обеспечивая основу для современных сервисно-ориентированных архитектур.

4.4. Подробности реализации

BI-SNAP[9] — это национальный стандарт, установленный Банком Индонезии с целью регулирования взаимодействия между поставщиками услуг платёжных систем [4]. Этот стандарт применяется не только к банковским учреждениям, но также является обязательным для небанковских организаций, таких как PSP[10], включая финтех-компании и стартапы, работающие в экосистеме цифровых платежей Индонезии.

Одним из ключевых компонентов BI-SNAP является внедрение протокола безопасности OAuth 2.0. OAuth 2.0 используется как стандарт авторизации, обеспечивающий безопасный, аутентифицированный доступ между системами, соответствующий правам доступа, предоставленным пользователем. Цель внедрения OAuth 2.0 в рамках BI-SNAP — обеспечение конфиденциальности данных клиентов, предотвращение несанкционированного доступа, а также упрощение интеграции между участниками платёжной индустрии.

Ниже описан процесс реализации OAuth 2.0 в BI-SNAP, как он применяется в рамках E-money Core System. На начальном этапе B2B-клиент например, бизнес-партнёр или сторонняя система отправляет запрос на авторизацию на OAuth-сервер. Получив запрос, сервер производит проверку учетных данных, чтобы убедиться, что запрашивающий клиент зарегистрирован и имеет официальное разрешение на сотрудничество в рамках экосистемы BI-SNAP. Если клиент подтверждён и признан действительным, OAuth-сервер обрабатывает запрос в соответствии с протоколом OAuth 2.0 и в ответ выдаёт токен доступа [3].

Этот токен может быть использован клиентом для получения доступа к определённым ресурсам или сервисам, предоставляемым системой E-money Core System, в рамках согласованных прав доступа . Для большей наглядности ниже приведена диаграмма последовательности, иллюстрирующая процесс отправки B2B-запроса клиентом на сервер авторизации.

[9]National Standard for Open API Payments

[10]Payment Service Providers

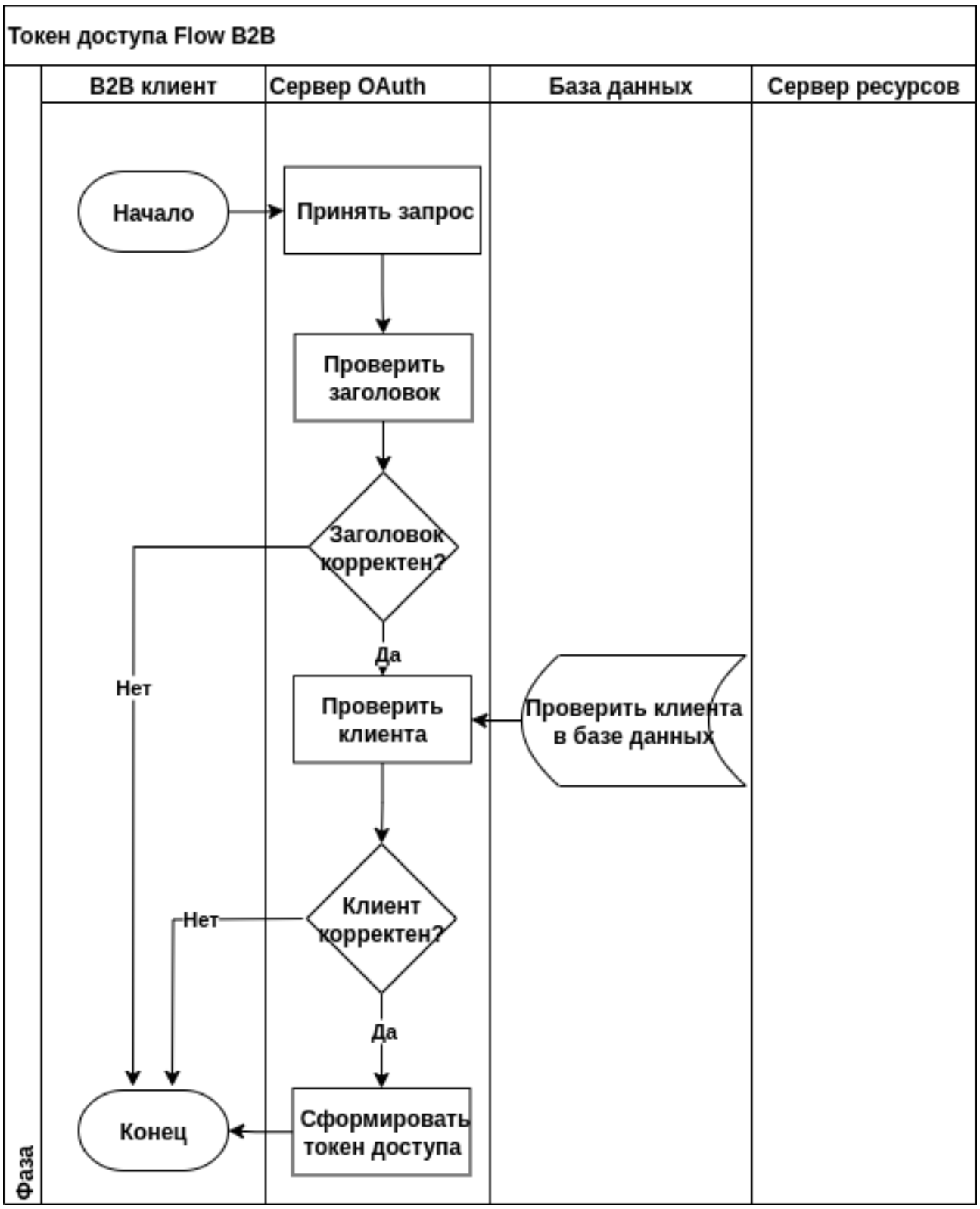

Рис. 2: Токен доступа B2B

Следующий процесс описывает, как в схеме B2B2C[11] осуществляется запрос токена доступа для получения доступа к серверу ресурсов. По-

[11]Business to Business to Consumer

сле того как B2B-клиент получает токен доступа, этот токен не используется напрямую конечным пользователем для обращения к серверу ресурсов. Вместо этого каждый клиент, зарегистрированный в системе B2B-клиента, должен пройти дополнительную процедуру аутентификации, чтобы получить собственный токен доступа от своего имени.

Другими словами, даже если B2B-клиент уже прошёл проверку и получил токен доступа, конечный пользователь всё равно должен отправить запрос авторизации через B2B-клиента. В этом процессе B2B-клиент выступает как посредник и перенаправляет запрос на OAuth-сервер. Если запрос считается допустимым и соответствует согласованной области доступа , OAuth-сервер выдаёт персонализированный токен доступа, предназначенный конкретному клиенту.

Затем этот токен, связанный с конкретным клиентом, используется для доступа к серверу ресурсов. Таким образом, каждый запрос к данным или сервисам является персонализированным и проходит индивидуальную проверку, соответствующую личности клиента и его правам доступа. Ниже приведена схема потока B2B2C для получения токена доступа.

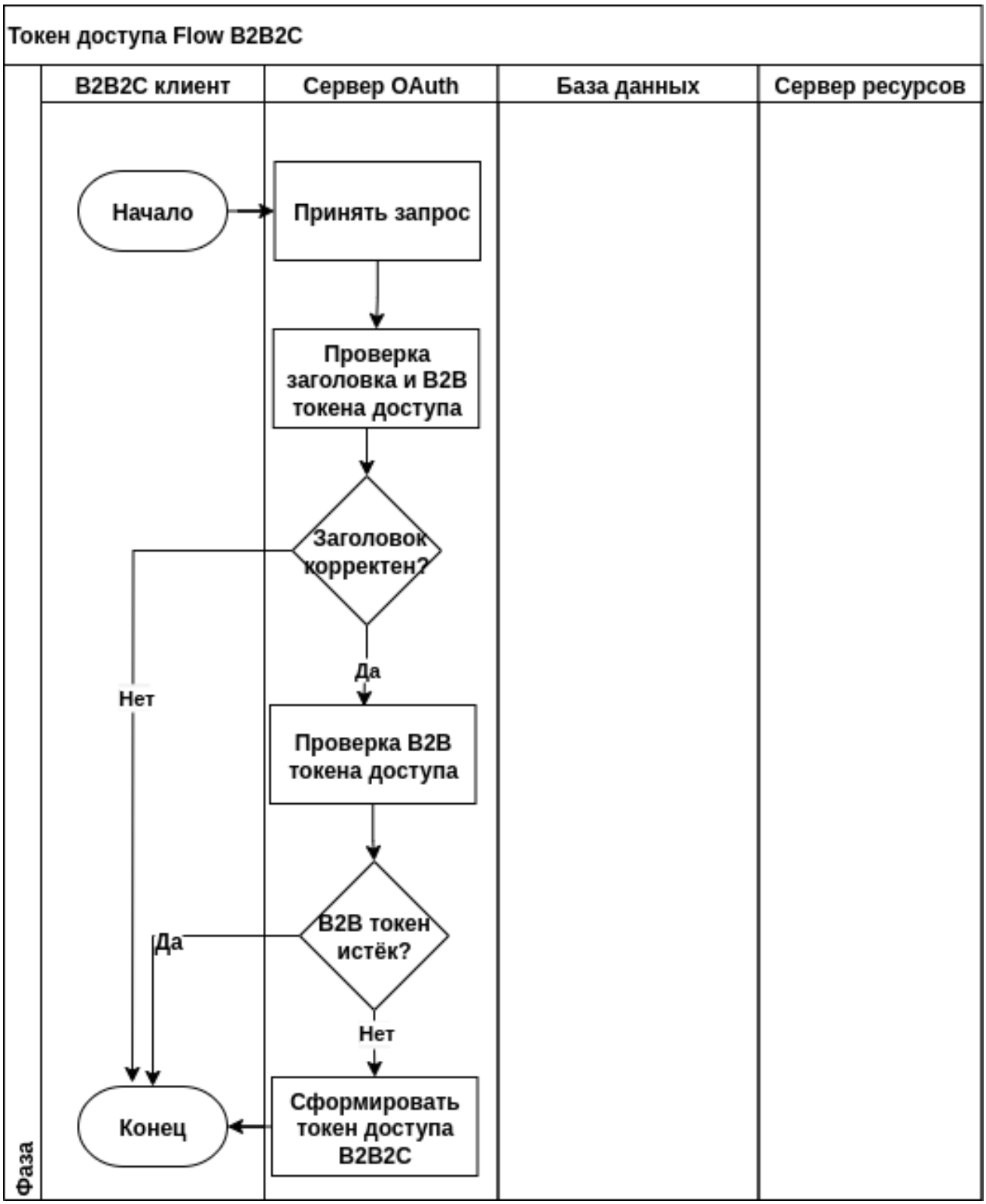

Рис. 3: Токен доступа B2B2C

Следующий шаг — это то, как пользователь может получить доступ к серверу ресурсов после того, как запросы на получение токенов доступа B2B и B2B2C были успешно выполнены. Ниже представлен полный

поток того, как схема B2B2C может получить доступ к серверу ресурсов.

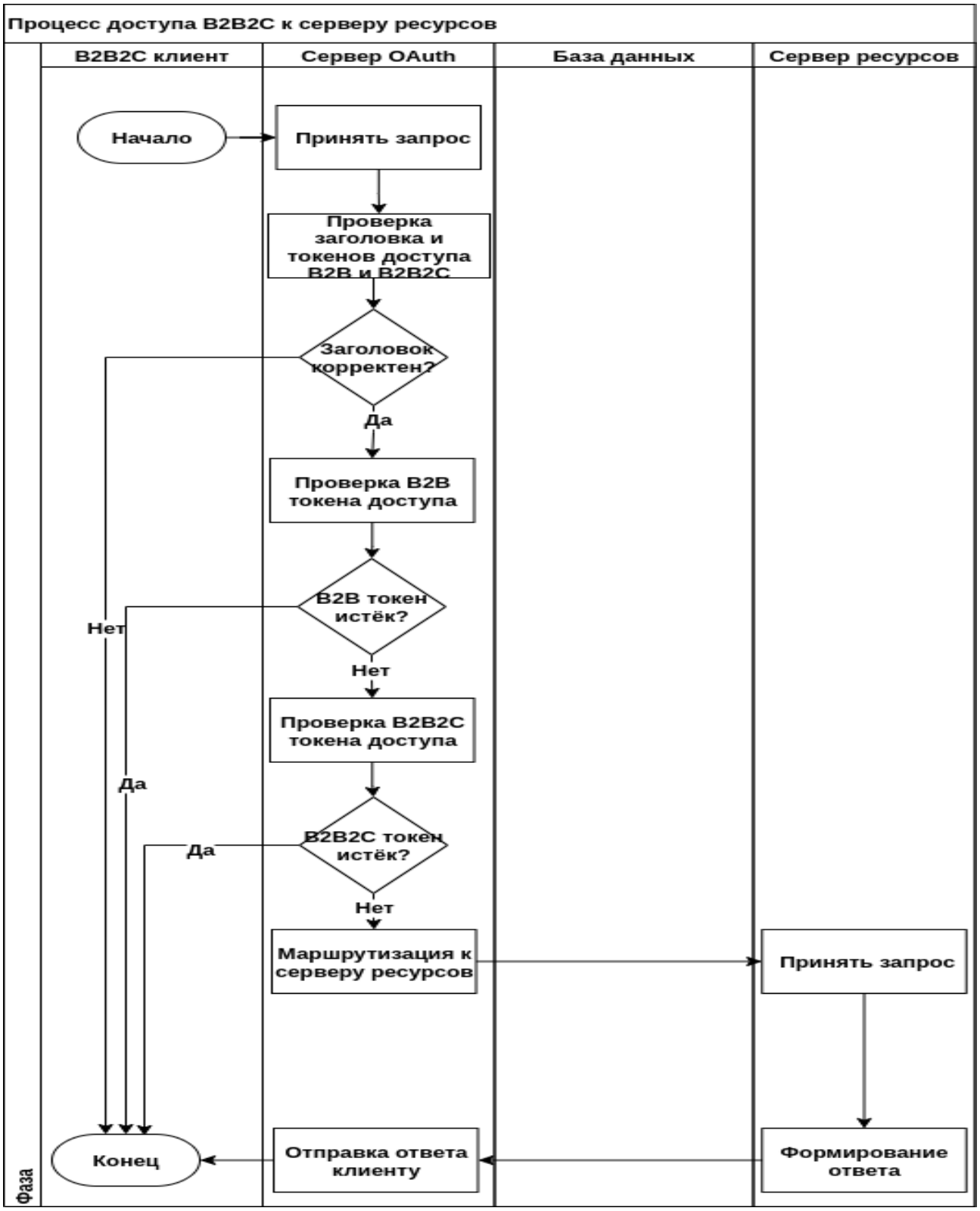

Рис. 4: Доступ B2B2C к серверу ресурсов

4.5. Проектирование базы данных

Для соблюдения положений и требований BI-SNAP необходимо разработать модель базы данных, которая сможет поддерживать процессы аутентификации и авторизации безопасным и структурированным образом. Одним из ключевых элементов в данной реализации является хранение информации, связанной с B2B-клиентами, которые официально заключили партнёрское соглашение. Цель заключается в том, чтобы отфильтровывать каждый входящий запрос к серверу OAuth и обеспечивать доступ к сервисам только зарегистрированным и авторизованным субъектам.

Благодаря такому механизму система может предотвращать несанкционированный доступ со стороны клиентов, которые не имеют партнёрских отношений или не соответствуют требованиям интеграции с основной системой E-money. Эта проверка осуществляется на основе данных, хранящихся в базе данных, что позволяет сохранять целостность и безопасность процесса авторизации. Кроме того, необходима таблица для хранения информации о правах доступа, предоставленных зарегистрированным B2B-клиентам. Для наглядности ниже приведено моделирование базы данных.

Рис. 5: Проектирование базы данных BI SNAP

Заключение

Результаты работы, выполненной в ходе учебной практики, включают завершение следующих задач:

- Проектирование базы данных.
- Реализация системы с использованием вышеупомянутых библиотек.
- Система OAuth 2 была развернута в промышленной среде компанией PT Duta Teknologi Kreatif.

С завершением проекта по внедрению OAuth 2 в соответствии со спецификациями BI SNAP обеспечивается высокий уровень безопасности в процессе аутентификации и обмена данными между системами.

Список литературы

[1] Google. Why and when to use API keys. — URL: https://cloud.google.com/endpoints/docs/openapi/when-why-api-key (дата обращения: 9 июля 2025 г.).

[2] Hardt Dick. The OAuth 2.0 Authorization Framework. — URL: https://datatracker.ietf.org/doc/html/rfc6749 (дата обращения: 2 июня 2025 г.).

[3] Indonesia Asosiasi Sistem Pembayaran. SNAP Dev Portal Documentation. — URL: https://apidevportal.aspi-indonesia.or.id/api-services/keamanan (дата обращения: 4 августа 2025 г.).

[4] Indonesia Bank. National Open API Payment Standard (SNAP). — URL: https://www.bi.go.id/en/layanan/standar/snap/default.aspx (дата обращения: 6 августа 2025 г.).

[5] Jetty Eclipse. HTTP Client. — URL: https://jetty.org/docs/jetty/12/programming-guide/client/http.html (дата обращения: 27 июля 2025 г.).

[6] Microsoft. What Is SAML (Security Assertion Markup Language)? — URL: https://www.microsoft.com/en-us/security/business/security-101/what-is-security-assertion-markup-language-saml (дата обращения: 1 июля 2025 г.).

[7] Oracle. Chapter 10 Java Servlet Technology. — URL: https://docs.oracle.com/cd/E19798-01/821-1841/bnafd/index.html (дата обращения: 25 июля 2025 г.).

[8] Wikipedia. History of the Internet. — URL: https://en.wikipedia.org/wiki/History_of_the_Internet (дата обращения: 20 июня 2025 г.).

[9] jwt.io. Introduction to JSON Web Tokens. — URL: https://www.jwt.io/introduction#what-is-json-web-token (дата обращения: 20 июля 2025 г.).