**Задание: Миграция авторизации с одиночного JWT на Access и Refresh токены**

**Цель задания**

* Освоить концепцию разделения токенов на Access (короткоживущий, для аутентификации) и Refresh (долгоживущий, для обновления Access токена).
* Реализовать обновлённую архитектуру авторизации, включая хранение сессий в базе данных с оптимистичной блокировкой.
* Добавить проверку типа токена с помощью дополнительного claim, чтобы гарантировать, что только access токены используются для аутентификации в фильтре.

1. **Теоретическая база**

В традиционной реализации авторизации с использованием одного JWT токена существует несколько важных недостатков. Если такой токен скомпрометирован, злоумышленник получает полный доступ к системе до истечения срока его действия. Кроме того, отсутствует механизм обновления токена без повторного прохождения полной аутентификации, что создаёт дополнительные неудобства для пользователя и снижает уровень безопасности.

Чтобы решить эти проблемы, современные системы авторизации всё чаще используют разделение токенов на два типа: Access и Refresh. Access токен предназначен для аутентификации в каждом запросе, имеет короткий срок жизни и минимизирует риск, связанный с его компрометацией. В случае, если он станет известен злоумышленнику, его использование будет ограничено по времени. Refresh токен, напротив, обладает более длительным сроком действия и используется исключительно для обновления Access токена через специальный эндпоинт. Такой подход позволяет пользователю продолжать работу без повторного ввода учетных данных, а системе – обеспечить дополнительный контроль за сессией.

Особое внимание уделяется привязке токенов к конкретным устройствам. Каждый комплект Access и Refresh токенов выдается для определённой сессии, связанной с уникальным идентификатором устройства (deviceId). Это позволяет отслеживать активные сессии, а при обнаружении подозрительной активности (например, повторное использование refresh токена) оперативно блокировать все сессии пользователя.

Хранение сессий в базе данных дополнительно повышает безопасность системы, однако при таком подходе может пострадать производительность. Здесь каждая сессия фиксируется с информацией о пользователе, устройстве, сроках действия токенов и их статусе.

**2. Архитектурные изменения и схемы**

**2.1. Общая архитектура системы**

Базовая схема работы: при логине пользователь проходит аутентификацию, после чего создаётся пара токенов (Access и Refresh) с привязкой к deviceId. Сессия сохраняется в базе данных.

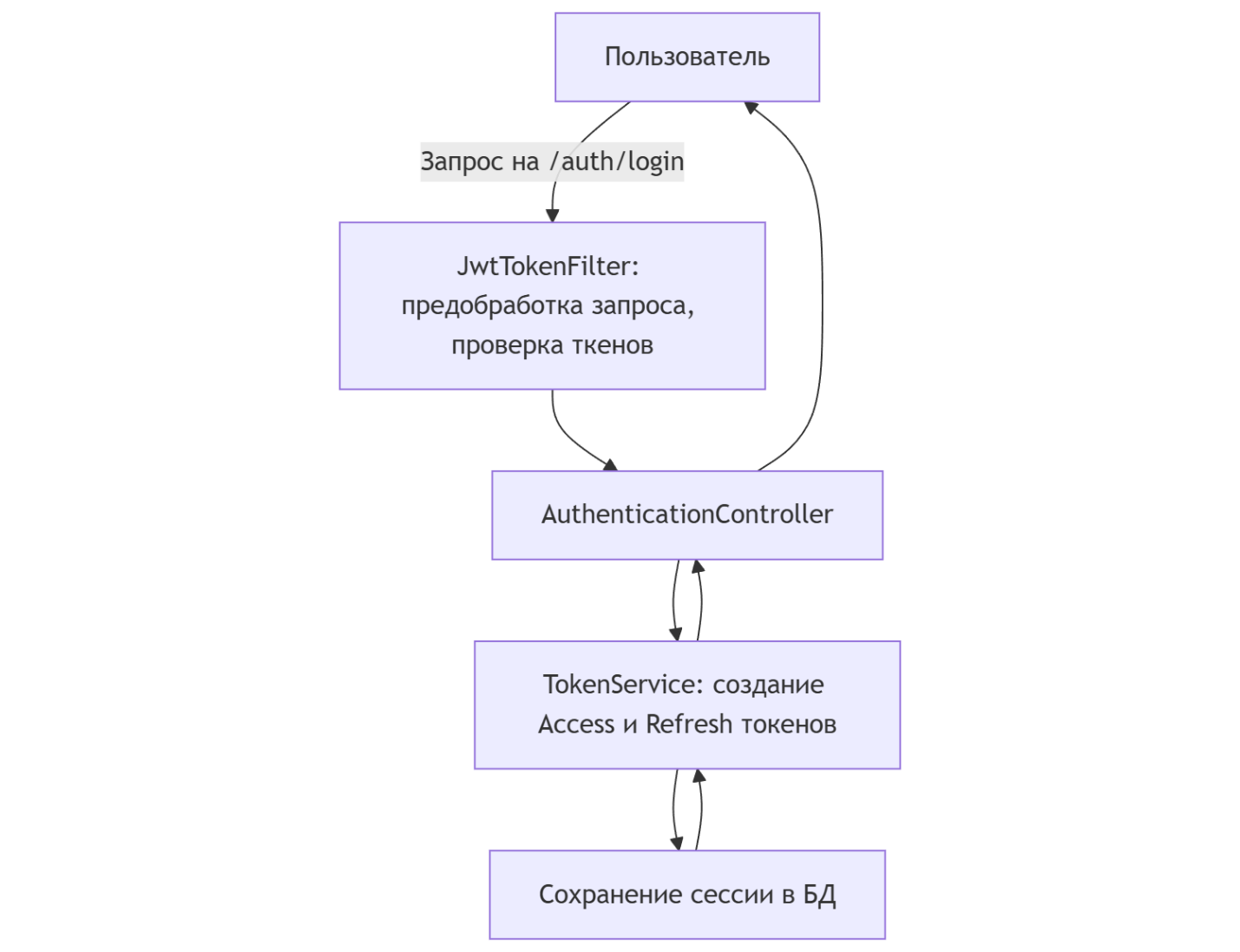


Рисунок 1. Общая схема работы

**2.2. Процесс обновления токенов**

При обращении на эндпоинт refresh происходит поиск сессии в БД. Если сессия соответствует условиям, старый refresh токен помечается как использованный и выдаётся новая пара токенов. Если условия не выполнены, система блокирует все сессии пользователя.

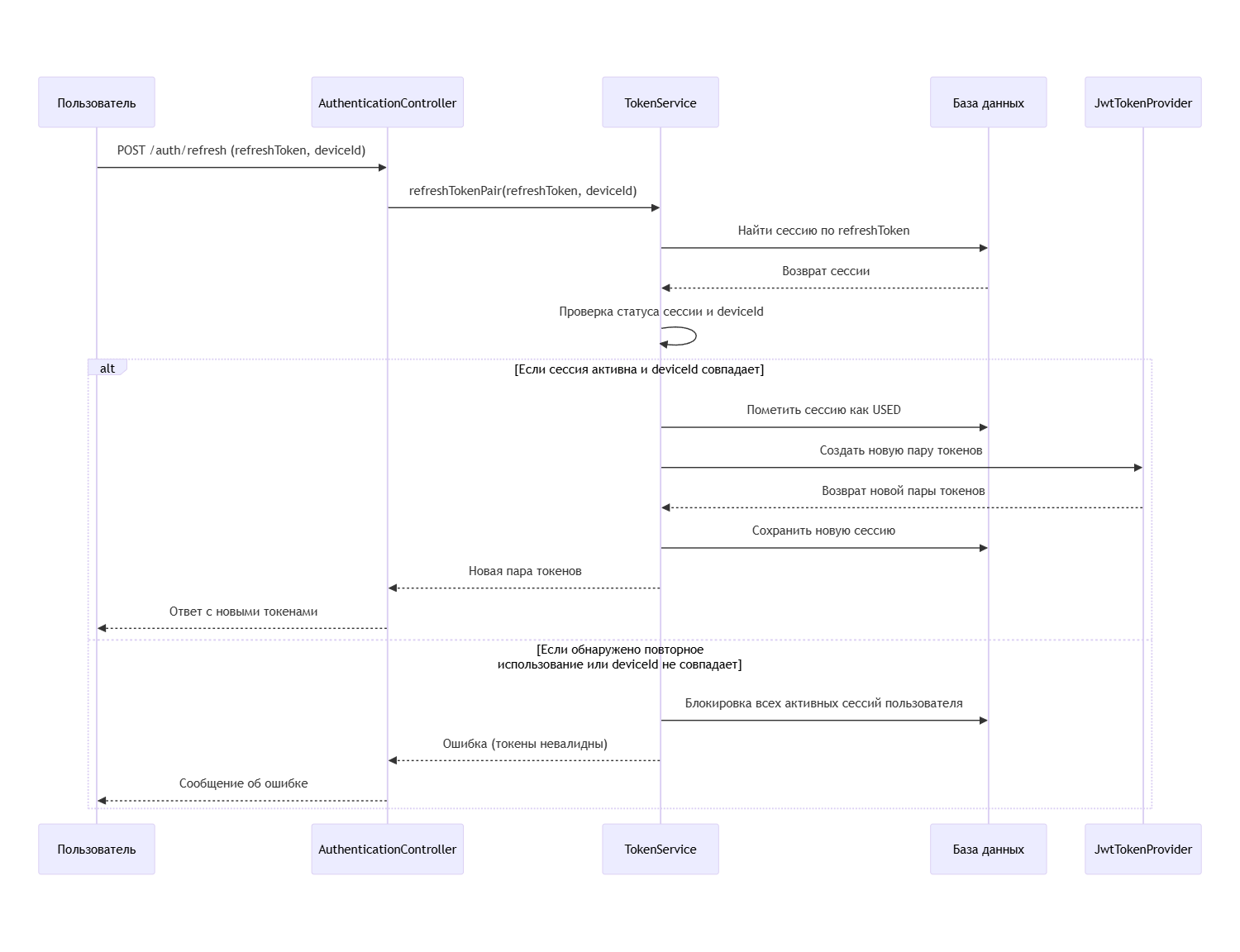
****

Рисунок 2. Диаграмма последовательности процесса обновления токенов

**2.3. Модификации в JWT фильтре**

Чтобы обеспечить безопасность, в каждом токене добавляется дополнительный claim token\_type. В фильтре проверяется, что переданный токен является именно access-токеном. Это предотвращает ошибочное использование refresh токена для аутентификации. Фильтр проверяет валидность токена, затем извлекает claim token\_type. Если тип равен «access», происходит установка аутентификации, иначе – запрос не пропускается.

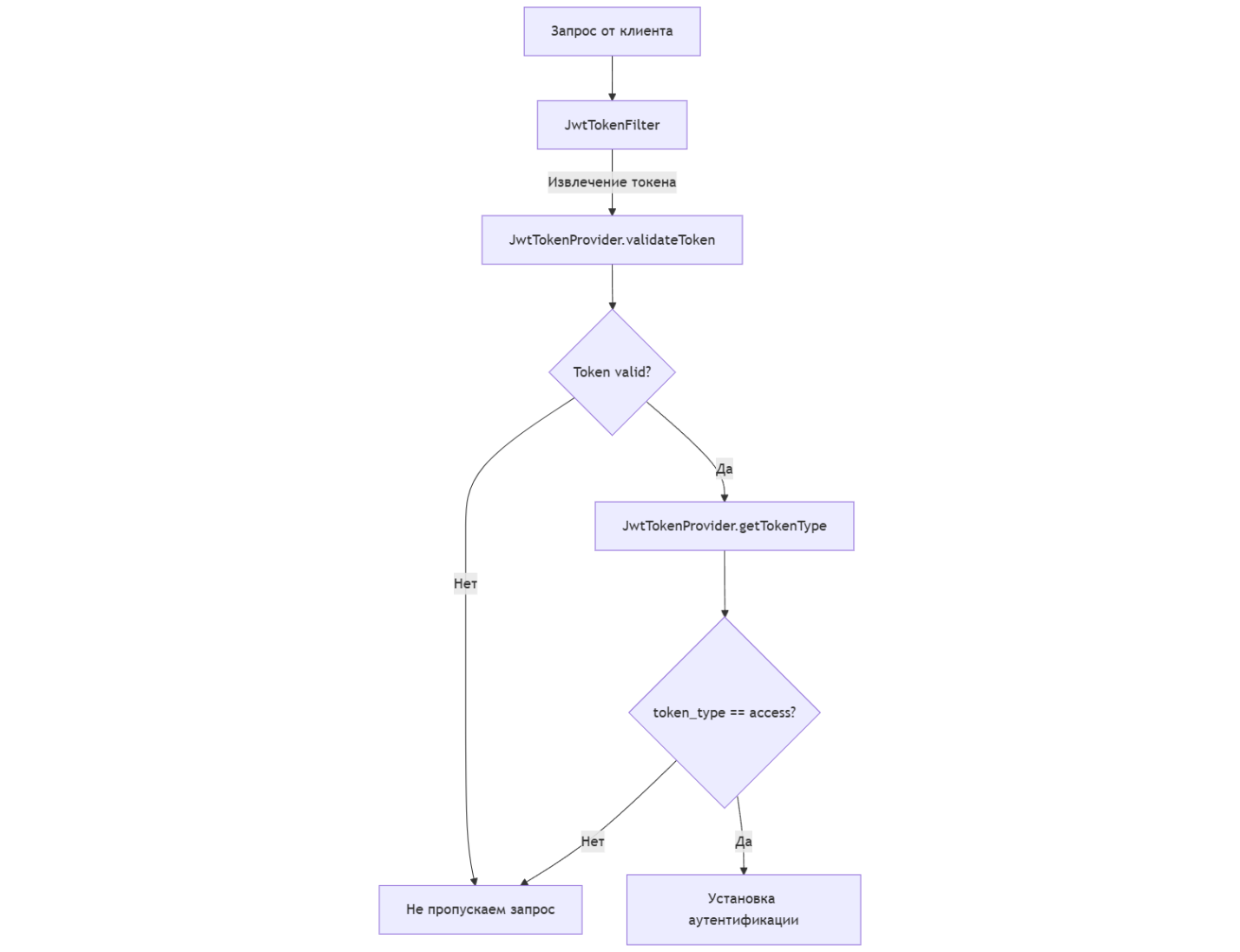


Рисунок 3. Схема проверки типа токена в фильтре

**2.4. Использование блокировок в базе данных**

В текущей реализации системы явных механизмов блокировок при обновлении записей в таблицах не используется. Однако при разработке многопоточных или распределённых приложений важно учитывать возможность возникновения конфликтов при одновременном обновлении одних и тех же данных.

Существует два основных подхода к управлению конкурирующим доступом к данным:

• Пессимистическая блокировка предполагает, что запись блокируется на уровне базы данных сразу при её чтении или перед обновлением, чтобы другие транзакции не могли её изменить до завершения текущей операции. Такой подход полезен, если вероятность конфликтов высока. Но он может негативно сказаться на производительности из-за удержания блокировок.

• Оптимистическая блокировка не использует явное блокирование записей. Вместо этого система отслеживает изменения данных с помощью специального поля, обычно версии, и при сохранении проверяет, что данные не были изменены другой транзакцией. Если обнаруживается расхождение, обновление отменяется, что позволяет корректно обработать конфликт. Этот метод эффективен, когда конфликты редки.

В нашем задании мы будем применять оптимистическую блокировку для работы с таблицей сессий, поскольку обновления сессий происходят нечасто, и вероятность возникновения конфликтов достаточно низка. Для реализации оптимистической блокировки в сущности, описывающей сессию, мы можем использовать аннотацию @Version над соответствующим полем. Эта аннотация автоматически управляет полем версии, которое увеличивается при каждом обновлении записи. Если два потока одновременно пытаются изменить одну и ту же запись, то несовпадение версий приведёт к возникновению исключения, что сигнализирует о конфликте обновлений и позволяет принять корректирующие меры.

**3. Практическая реализация**

1. **Модифицировать JwtTokenProvider:**

Добавить методы для создания двух типов токенов:

* createAccessToken – включает claim "token\_type": "access".
* createRefreshToken – включает claim "token\_type": "refresh" и привязывает refresh токен к deviceId.
* Добавить метод getTokenType, возвращающий значение claim "token\_type".

1. **Модифицировать JwtTokenFilter:**

После проверки валидности токена, извлекать тип токена. Устанавливать аутентификацию только для access токена (если token\_type равен "access").

1. **Реализовать сущность UserSession и репозиторий:**

* Создать класс UserSession для хранения данных сессии (email, deviceId, accessToken, refreshToken, сроки истечения, статус, версия).
* Создать enum SessionStatus с состояниями (ACTIVE, USED, REVOKED).
* Реализовать UserSessionRepository для доступа к данным сессий.

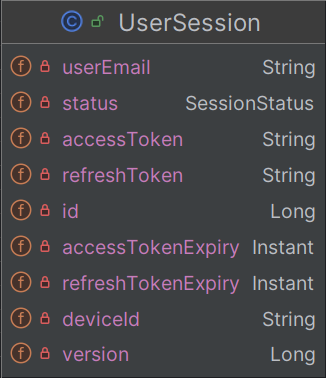


Рисунок 4. Возможный вариант структура таблицы UserSession

1. **Создать TokenService:**

Реализовать методы:

* issueTokenPair для генерации новой пары токенов и сохранения сессии.
* refreshTokenPair для обновления пары токенов с проверкой сессии и блокировкой всех сессий в случае нарушения.
* blockAllSessionsForUser для блокировки сессий пользователя.

1. **Модифицировать AuthenticationController:**

* Изменить эндпоинт /auth/login так, чтобы он принимал параметр deviceId и возвращал пару токенов.
* Добавить новый эндпоинт /auth/refresh для обновления токенов с использованием refresh-токена.

1. **Тестирование и отладка:**

Запустить приложение и протестировать логику аутентификации и проверить, что при использовании refresh токена с неправильным deviceId или повторном использовании происходит блокировка сессий.