# Архитектурная Трансформация: Теоретические Основы Инфраструктуры Идентификации и Безопасности (День 1)

## Аннотация

Настоящий отчет представляет собой исчерпывающее теоретическое руководство, разработанное в рамках программы интенсивной переквалификации Unity-разработчиков в инженеров Enterprise.NET Backend. Документ фокусируется на фундаментальных аспектах первого дня модуля "Инфраструктура Идентификации", анализируя смену архитектурных парадигм, механику протоколов OAuth 2.0 и OpenID Connect, криптографические стандарты JWT, а также специфику реализации сервера авторизации на базе фреймворка OpenIddict. Цель документа — сформировать глубокое понимание процессов обеспечения безопасности в распределенных системах, необходимое для реализации финального проекта (Capstore Project) — Системы Бронирования Переговорных Комнат.1

## 1. Эпистемологический Сдвиг: От Игрового Цикла к Распределенным Транзакциям

### 1.1. Деконструкция Ментальных Моделей Unity-разработчика

Переход из среды разработки игр (GameDev) в сферу корпоративного бэкенда (Enterprise Backend) требует не просто изучения новых синтаксических конструкций C#, но и фундаментальной перестройки мышления. Unity-инженер привык существовать в детерминированном мире "Игрового Цикла" (Game Loop). В этой реальности время дискретно и измеряется кадрами (Update, FixedUpdate), а состояние мира (позиции игроков, инвентарь) находится в оперативной памяти, доступной мгновенно.1

В контексте сетевого взаимодействия Unity-разработчики часто оперируют моделями, основанными на постоянных соединениях (Stateful). Протоколы транспортного уровня, используемые в играх (TCP сокеты, RUDP, библиотеки типа Photon или Mirror), создают иллюзию непрерывного канала связи. После того как игрок прошел аутентификацию (часто неявно, через SteamID или DeviceID), сервер поддерживает контекст этого соединения. Переменная Client.Connection является гарантом идентичности: пока сокет открыт, сервер "знает", кто находится на другом конце провода.

### 1.2. Реальность Enterprise Backend: Stateless и Асинхронность

Корпоративные системы, построенные на архитектуре REST или gRPC, функционируют в парадигме отсутствия состояния (Stateless). Это означает, что сервер не хранит информацию о предыдущих запросах клиента между транзакциями. Каждый HTTP-запрос — это изолированное событие, "чистый лист", который должен содержать всю необходимую информацию для его обработки, включая доказательство идентичности инициатора.2

Для Unity-разработчика это создает ряд когнитивных диссонансов:

1. **Потеря "Соседства"**: Объекты больше не живут в одном адресном пространстве. Обращение к соседнему сервису — это не вызов метода, а сетевой запрос, подверженный задержкам, сбоям и таймаутам.
2. **Асинхронность I/O**: Если в Unity блокировка основного потока приводит к падению FPS, то в бэкенде блокировка потока пула (Thread Pool) при синхронном ожидании ввода-вывода (I/O) приводит к исчерпанию ресурсов сервера (Thread Starvation) и отказу в обслуживании тысяч пользователей.1
3. **Явная Идентификация**: Нельзя полагаться на открытый сокет. Каждый запрос к API бронирования должен сопровождаться цифровым "паспортом" — токеном безопасности.

### 1.3. Архитектурные Императивы Перехода

В рамках учебного плана "Неделя 14" ставится задача спроектировать систему, соответствующую промышленным стандартам надежности и безопасности.1 Это диктует отказ от упрощенных моделей ("монолитная аутентификация") в пользу федеративной идентичности. Мы больше не храним пароли в таблице Users основного приложения. Мы делегируем ответственность за проверку паролей специализированному сервису — Провайдеру Идентичности (Identity Provider, IdP).

Эта модель решает две критические задачи:

* **Безопасность**: Централизация управления учетными данными снижает поверхность атаки. Если нужно отключить пользователя, это делается в одной точке.
* **Масштабируемость**: Сервис бронирования (Booking Service) не тратит ресурсы на хэширование паролей; он лишь проверяет криптографическую подпись токена, что является быстрой операцией CPU.3

### Таблица 1.1: Сравнительный анализ парадигм

| **Характеристика** | **Unity / GameDev (Stateful)** | **Enterprise Backend (Stateless)** | **Архитектурное следствие для проекта** |
| --- | --- | --- | --- |
| **Жизненный цикл** | Синхронный, Frame-based (кадры). | Асинхронный, Request-Response. | Использование async/await для всех операций I/O, освобождение ThreadPool.1 |
| **Идентичность** | Привязана к сессии/сокету. | Передается с каждым запросом (Токен). | Внедрение OAuth 2.0 и JWT Bearer Authentication. |
| **Безопасность** | Платформенная (Steam, PSN). | Федеративная (OIDC). | Использование OpenIddict для реализации IdP. |
| **Масштабирование** | Вертикальное (оптимизация кода). | Горизонтальное (репликация инстансов). | Вынос состояния сессий и кэшей в Redis.1 |

## 2. Теория Идентичности: Протоколы OAuth 2.0 и OpenID Connect

### 2.1. Дихотомия Авторизации и Аутентификации

Фундаментальной ошибкой начинающих бэкенд-инженеров является смешение понятий Аутентификации (Authentication — AuthN) и Авторизации (Authorization — AuthZ). В Enterprise-системах это различие кодифицировано в используемых протоколах.

* **OAuth 2.0 (Open Authorization)** — это протокол **авторизации**. Его цель — ответить на вопрос: *"Что этот субъект имеет право делать?"*. Он был создан для решения проблемы делегирования доступа. Классический пример: вы хотите позволить сервису печати фотографий доступ к вашим фото в Google Drive, не передавая сервису печати ваш пароль от Google. OAuth 2.0 выдает "парковочный талон" (Access Token), который позволяет сервису взять машину (фотографии), но не дает права продать её или зайти в бардачок (читать почту).3
* **OpenID Connect (OIDC)** — это протокол **аутентификации**, надстроенный поверх OAuth 2.0. Его цель — ответить на вопрос: *"Кто этот субъект?"*. OAuth 2.0 сам по себе ничего не знает о пользователе; для него токен — это просто непрозрачная строка прав. OIDC добавляет стандартизацию, вводя понятие **ID Token** — структурированного документа (JWT), который содержит информацию о профиле пользователя (имя, email, фото).6

В контексте нашей Системы Бронирования:

1. **OIDC** работает, когда пользователь входит в Unity-клиент или веб-панель. Приложение получает id\_token, декодирует его и пишет: "Добро пожаловать, Иван".
2. **OAuth 2.0** работает, когда приложение отправляет запрос POST /api/bookings. Приложение прикладывает access\_token, и API проверяет, есть ли у токена право (Scope) на создание брони.

### 2.2. Ролевая Модель Протокола

Для корректной реализации OpenIddict необходимо четко понимать роли участников обмена:

1. **Resource Owner (Владелец Ресурса)**: Обычно это человек, конечный пользователь, который управляет своими данными (календарем бронирований).4
2. **User-Agent**: Браузер или Unity-клиент, через который пользователь взаимодействует с системой.
3. **Client (Клиент)**: Приложение, которое запрашивает доступ к ресурсам от имени пользователя. Важно: в терминологии OAuth "Клиент" — это программный код (наш Unity App или React SPA), а не человек.7
4. **Authorization Server (Сервер Авторизации)**: Сервис, который проверяет учетные данные пользователя (логин/пароль) и выдает токены безопасности. В нашем проекте эту роль играет приложение ASP.NET Core с интегрированным OpenIddict.3
5. **Resource Server (Сервер Ресурсов)**: API, которое хранит защищенные данные (Booking Service). Оно принимает Access Token и принимает решение о допуске к данным, но само токены не выдает.7

### 2.3. Эволюция Потоков (Flows): Смерть Implicit Flow

История развития OAuth 2.0 — это история борьбы с уязвимостями. Понимание того, *почему* определенные методы стали запрещенными, критически важно для архитектора безопасности.

#### 2.3.1. Анатомия Implicit Flow (Устаревший)

Изначально для SPA (Single Page Applications), работающих в браузере, был разработан **Implicit Flow** (Неявный поток). Поскольку JavaScript в браузере не мог безопасно хранить секреты и кросс-доменные запросы (CORS) были сложны, этот поток предполагал упрощенную схему:

1. Пользователь перенаправляется на сервер авторизации.
2. После логина сервер перенаправляет пользователя обратно, вставляя access\_token прямо во фрагмент URL (после #): https://myapp.com/callback#access\_token=eyJ....
3. JS-скрипт на странице считывает токен из URL и использует его.

#### 2.3.2. Векторы Атаки и Депрекация

Современный анализ безопасности (OAuth 2.1 drafts, IETF BCP) признал Implicit Flow категорически небезопасным и подлежащим исключению.8

Основные причины:

* **Утечка через Referer Header**: Если сразу после получения токена пользователь перейдет по ссылке на сторонний сайт, браузер может передать полный URL (включая токен) в заголовке Referer. Злоумышленник, владеющий сторонним сайтом, получит токен в логах своего сервера.12
* **История Браузера**: URL с токеном сохраняется в истории браузера, что делает его доступным для любого, кто получит доступ к устройству.12
* **Access Token Injection**: В Implicit Flow нет механизма, гарантирующего, что токен был выдан именно тому клиенту, который его запросил. Злоумышленник может подсунуть свой токен жертве.14

### 2.4. Современный Стандарт: Authorization Code Flow с PKCE

На смену Implicit Flow пришел **Authorization Code Flow с расширением PKCE (Proof Key for Code Exchange)**. Изначально разработанный для мобильных приложений (которые не могут хранить Client Secret), он стал золотым стандартом для всех публичных клиентов, включая SPA и Unity WebGL.15

Механика PKCE (произносится "пикси") добавляет криптографическую защиту от перехвата кода авторизации. Даже если злоумышленник перехватит временный код, он не сможет обменять его на токен.

#### 2.4.1. Алгоритм PKCE: Математика Защиты

Процесс обмена состоит из двух фаз и использует одностороннюю хэш-функцию SHA-256.

1. **Инициализация (на Клиенте)**:
   * Клиент генерирует случайную строку высокой энтропии — **Code Verifier**. Это криптографический секрет сессии.
   * Клиент вычисляет хэш от этой строки: Code Challenge = Base64Url(SHA256(Code Verifier)).19
   * *Пример*: Если Verifier = "dBjftJeZ4CVP-mB92K27uhbUJU1p1r\_wW1gFWFOEjXk", то Challenge (S256) = "E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM".19
2. **Запрос Авторизации (Front-Channel)**:
   * Клиент перенаправляет пользователя на /authorize, передавая client\_id и code\_challenge (но не Verifier!).
   * Сервер авторизации сохраняет code\_challenge и ассоциирует его с выдаваемым authorization\_code.
3. **Обмен Кода (Back-Channel)**:
   * Клиент получает код и делает прямой POST-запрос на /token. В этом запросе он передает authorization\_code и исходный Code Verifier.
4. **Верификация (на Сервере)**:
   * Сервер берет полученный Code Verifier, применяет к нему SHA-256 и сравнивает результат с сохраненным ранее code\_challenge.
   * SHA256(Verifier\_from\_request) == Challenge\_from\_memory.
   * Если они совпадают, сервер гарантированно знает, что запрос на обмен кода пришел от того же приложения, которое начинало процесс. Перехват кода "посередине" становится бесполезным, так как злоумышленник не знает Code Verifier (восстановить его из хэша невозможно).15

## 3. Криптографическая Анатомия: JSON Web Token (JWT)

В экосистеме.NET и OpenIddict основным носителем информации об идентичности является JWT (RFC 7519). Понимание его структуры обязательно для отладки интеграции между микросервисами.

### 3.1. Трехчастная Структура

JWT представляет собой строку, состоящую из трех сегментов, разделенных точкой (.): Header.Payload.Signature. Каждый сегмент закодирован в Base64Url.

#### 3.1.1. Header (Заголовок)

Содержит метаданные о алгоритмах криптографии.

JSON

{  
 "alg": "RS256",  
 "typ": "JWT",  
 "kid": "12345..."  
}

* alg: Алгоритм подписи. RS256 (RSA Signature with SHA-256) является стандартом для OIDC. Он асимметричен: Auth Service подписывает приватным ключом, а все остальные (Resource Servers) проверяют публичным.
* kid (Key ID): Идентификатор ключа. Критически важен для ротации ключей (Key Rollover). Когда Auth Service меняет ключи, Resource Server использует kid, чтобы найти правильный публичный ключ в наборе JWKS.23

#### 3.1.2. Payload (Полезная Нагрузка)

Содержит утверждения (Claims) — факты о пользователе и токене.

* **Стандартные клеймы (Registered Claims)**:
  + iss (Issuer): Кто выдал токен. (напр. https://auth.mysystem.com). Resource Server обязан проверять это поле, чтобы не принять токен от чужого сервера.25
  + sub (Subject): Уникальный идентификатор пользователя. Это **единственное** поле, которое гарантированно идентифицирует пользователя стабильно во времени. Email может измениться, но sub (обычно GUID) остается неизменным.26
  + aud (Audience): Для кого предназначен токен (напр. booking-api). Если сервис получает токен, где в aud не указано его имя, он должен отклонить запрос с ошибкой 401. Это защита от переиспользования токенов в несвязанных сервисах.25
  + exp (Expiration Time): Время истечения (Unix timestamp).
  + iat (Issued At): Время выдачи.
* **Кастомные клеймы**: Роли (role), email (email), имя (name).29

#### 3.1.3. Signature (Подпись)

Создается путем хэширования заголовка и полезной нагрузки с использованием секретного ключа. Подпись гарантирует целостность данных: если злоумышленник изменит в Payload role: "user" на role: "admin", хэш перестанет совпадать с подписью, и токен будет отвергнут.30

### 3.2. JWS vs. JWE: Нюансы OpenIddict

Важнейший аспект, отличающий OpenIddict от многих других решений (например, Auth0), заключается в его подходе к приватности Access Token.

* **JWS (Signed)**: Токен подписан, но его содержимое (Payload) открыто для чтения (Base64 decode). Это стандартный подход для ID Token.
* **JWE (Encrypted)**: Токен не только подписан, но и полностью зашифрован. Его содержимое невозможно прочитать без приватного ключа дешифровки.

По умолчанию **OpenIddict шифрует Access Tokens**.31 Это сделано для того, чтобы клиенты (Frontend) не полагались на содержимое Access Token (который предназначен только для API) и чтобы скрыть внутреннюю структуру клеймов от посторонних глаз. Однако, это усложняет отладку на этапе разработки (нельзя вставить токен в jwt.io и посмотреть клеймы) и требует настройки обмена ключами шифрования между Auth Service и Resource Server.

В рамках первого дня обучения, для упрощения отладки и визуализации процессов, мы часто используем опцию DisableAccessTokenEncryption(), переводя токены в формат JWS.31

## 4. Архитектура OpenIddict: Отказ от Монолитных Решений

### 4.1. Сравнительный Анализ: IdentityServer vs OpenIddict

Долгое время стандартом де-факто в.NET был IdentityServer4. Однако его трансформация в коммерческий продукт Duende IdentityServer с лицензионными отчислениями для крупных компаний создала потребность в альтернативах.32

**OpenIddict** — это не готовый продукт "из коробки" (как Keycloak), а **фреймворк** для построения собственного сервера авторизации.

* **Гибкость**: Он не навязывает структуру базы данных или UI страниц логина. Вы сами пишете контроллеры и представления (Views), что позволяет полностью интегрировать процесс входа в дизайн вашего приложения.
* **Образовательная ценность**: OpenIddict работает на низком уровне абстракции, предоставляя события и менеджеры. Работа с ним заставляет разработчика глубоко понимать спецификацию протокола, так как многие вещи нужно конфигурировать явно.34

### 4.2. Компонентная Триада OpenIddict

Архитектура OpenIddict модульна и состоит из трех основных слоев:

1. **OpenIddict.Core**: Ядро системы. Отвечает за абстракции хранилища (Stores) и менеджеров (ApplicationManager, AuthorizationManager, ScopeManager). Оно не зависит от HTTP и может использоваться в любых типах приложений.
2. **OpenIddict.Server**: Компонент, реализующий протокол OpenID Connect Provider. Он перехватывает запросы к эндпоинтам (Token, Authorize, UserInfo), валидирует их, вызывает события и генерирует ответы. Интегрируется в конвейер ASP.NET Core.36
3. **OpenIddict.Validation**: Компонент для Resource Server. Он используется для проверки входящих токенов. В простых сценариях его можно заменить стандартным Microsoft.AspNetCore.Authentication.JwtBearer, но OpenIddict.Validation предлагает более глубокую интеграцию, например, валидацию Introspection (проверка статуса токена на сервере в реальном времени).37

## 5. Практическая Реализация Инфраструктуры (День 1)

### 5.1. Конфигурация Сервера Авторизации (Program.cs)

Настройка OpenIddict в.NET 8 производится через Fluent API в файле Program.cs. Это декларативное описание возможностей сервера.

#### 5.1.1. Регистрация Эндпоинтов и Потоков

Мы должны явно указать OpenIddict, какие URL будут использоваться для взаимодействия:

C#

services.AddOpenIddict()  
 .AddServer(options =>  
 {  
 // Установка эндпоинтов  
 options.SetAuthorizationEndpointUris("/connect/authorize")  
 .SetTokenEndpointUris("/connect/token")  
 .SetUserinfoEndpointUris("/connect/userinfo");  
  
 // Включение Authorization Code Flow с обязательным PKCE  
 options.AllowAuthorizationCodeFlow()  
 .RequireProofKeyForCodeExchange();  
   
 // Включение Refresh Token Flow для долгоживущих сессий  
 options.AllowRefreshTokenFlow();  
  
 // Регистрация скоупов (областей доступа)  
 options.RegisterScopes(OpenIddictConstants.Scopes.Email,   
 OpenIddictConstants.Scopes.Profile,   
 OpenIddictConstants.Scopes.Roles);  
   
 // Интеграция с ASP.NET Core  
 options.UseAspNetCore()  
 .EnableAuthorizationEndpointPassthrough()  
 .EnableTokenEndpointPassthrough()  
 .EnableUserinfoEndpointPassthrough();  
 });

Опция Enable...Passthrough() критически важна. Она говорит OpenIddict не обрабатывать запрос полностью самостоятельно, а передать управление в наш контроллер (например, AuthorizationController), где мы сможем реализовать кастомную логику показа страницы согласия (Consent Screen) или логина.36

#### 5.1.2. Управление Криптографическими Ключами

Для подписи токенов необходимы сертификаты.

* **Development Mode**: OpenIddict предоставляет методы AddDevelopmentSigningCertificate() и AddDevelopmentEncryptionCertificate(). Они генерируют самоподписанные сертификаты и сохраняют их локально.
* **Ephemeral Keys**: Метод AddEphemeralSigningKey() создает ключи в оперативной памяти. **Важно:** при перезапуске приложения (или контейнера Docker) ключи теряются, и все ранее выданные токены становятся невалидными. Это допустимо только для локальных тестов, но создает проблемы, если Auth Service перезагружается чаще, чем истекает срок жизни токенов.38

Для соответствия требованиям плана обучения (упрощение отладки JWT), мы отключаем шифрование Access токенов:

C#

options.DisableAccessTokenEncryption();

Это переводит их в формат JWS, читаемый на клиенте и в отладчике.31

### 5.2. Моделирование Клиентов и База Данных

OpenIddict использует Entity Framework Core для хранения информации о зарегистрированных приложениях (Clients) и авторизациях (Authorizations).

Для нашего проекта мы должны зарегистрировать клиента (Unity/React App) через IOpenIddictApplicationManager в Seeder-классе:

* **ClientId**: meeting-room-app
* **ClientType**: Public (так как клиент работает на устройстве пользователя и не может хранить секрет).
* **RedirectUris**:
  + http://localhost:3000/callback (для React SPA)
  + com.company.game://oauth/callback (для Unity Deep Linking).39
* **Permissions**: Явное перечисление разрешенных грантов (AuthorizationCode, RefreshToken) и скоупов.

### 5.3. Интеграция Клиента Unity и Deep Linking

Особое внимание в плане обучения уделяется интеграции с Unity. В отличие от браузера, Unity-приложение не имеет адресной строки.

#### 5.3.1. Deep Linking

Для завершения OAuth-потока на мобильном устройстве или десктопе используется механизм Deep Linking (Custom URL Schemes).

1. Unity открывает системный браузер по ссылке /connect/authorize.
2. Пользователь логинится на сайте Auth Service.
3. Auth Service редиректит на com.company.game://oauth/callback?code=....
4. Операционная система (Android/iOS/Windows), видя протокол com.company.game, запускает наше приложение и передает URL в качестве аргумента.
5. В Unity скрипт подписывается на событие Application.deepLinkActivated и извлекает code из URL.41

#### 5.3.2. Безопасность Хранения

Сохранение токенов в PlayerPrefs является грубой ошибкой безопасности, так как на Android/Windows это незашифрованный XML/Registry файл. В рамках курса мы рассматриваем использование безопасных хранилищ (Android Keystore / iOS Keychain) или хранение токенов исключительно в оперативной памяти (с обновлением через Refresh Token при старте).43

### 5.4. Настройка Resource Server (Валидация)

#### 5.4.1. OIDC Discovery

Resource Server (Booking API) должен доверять токенам Auth Service. Для этого он использует механизм OIDC Discovery.

Auth Service публикует документ метаданных по адресу /.well-known/openid-configuration. Этот JSON содержит:

* issuer: Имя издателя.
* jwks\_uri: Ссылка на публичные ключи (JWKS).  
  Resource Server при старте скачивает этот документ, затем скачивает ключи с jwks\_uri и использует их для проверки подписи входящих токенов.23

#### 5.4.2. Конфигурация JwtBearer

В Program.cs сервиса бронирования настраивается аутентификация:

C#

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)  
 .AddJwtBearer(options =>  
 {  
 options.Authority = "https://localhost:5001"; // Адрес Auth Service  
 options.Audience = "booking-api"; // Ожидаемый aud в токене  
 options.TokenValidationParameters.ValidateAudience = true;  
 options.TokenValidationParameters.ValidateIssuer = true;  
 });

Если Audience не настроен корректно на стороне выдачи токена (в Auth Service нужно явно прикрепить ресурс к токену через SetResources и SetDestinations), Resource Server отклонит токен с ошибкой 401.37

## 6. Управление Клеймами и Destinations

Одной из самых частых проблем при работе с OpenIddict является "пропажа" клеймов. Разработчик добавляет роль пользователю, но в Access Token она не приходит.

Причина кроется в механизме Destinations (Назначения). OpenIddict по умолчанию не включает клеймы (кроме sub) в Access Token, чтобы сохранить его размер минимальным.

В контроллере авторизации (AuthorizationController) необходимо явно указывать, куда должен попасть каждый клейм:

C#

foreach (var claim in principal.Claims)  
{  
 claim.SetDestinations(GetDestinations(claim, principal));  
}  
  
// Логика распределения  
private IEnumerable<string> GetDestinations(Claim claim, ClaimsPrincipal principal)  
{  
 // Имя и профиль идут в ID Token (для отображения на клиенте)  
 if (claim.Type == OpenIddictConstants.Claims.Name)  
 yield return OpenIddictConstants.Destinations.IdentityToken;  
  
 // Роли и права идут в Access Token (для проверки на API)  
 if (claim.Type == OpenIddictConstants.Claims.Role)  
 yield return OpenIddictConstants.Destinations.AccessToken;  
}

Без этого кода клеймы останутся только в билете аутентификации на сервере, но не попадут в JWT.46

## 7. Заключение

Первый день модуля закладывает фундамент безопасности всей системы. Мы перешли от интуитивной модели постоянного соединения Unity к строгой, криптографически верифицируемой модели федеративной идентичности. Использование OpenIddict вместо готовых решений дает нам полный контроль над потоками данных, позволяя реализовать сложные сценарии, такие как Deep Linking для мобильных клиентов и гранулярное управление содержимым токенов через Destinations. Эти знания являются базисом для следующих дней, где мы будем строить бизнес-логику бронирования, опираясь на доверенную идентификацию пользователей.

#### Источники

1. Неделя 14
2. Whats the difference between form based authentication Vs Oauth 2.0? - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/49892032/whats-the-difference-between-form-based-authentication-vs-oauth-2-0>
3. OAuth 2.0 and OpenID Connect overview - Okta Developer, дата последнего обращения: декабря 6, 2025, <https://developer.okta.com/docs/concepts/oauth-openid/>
4. OAuth 2.0 vs. OpenID Connect: Understanding Critical Differences - Avatier, дата последнего обращения: декабря 6, 2025, <https://www.avatier.com/blog/oauth-2-0-vs-openid/>
5. OAuth 2.1 vs 2.0: What developers need to know - Stytch, дата последнего обращения: декабря 6, 2025, <https://stytch.com/blog/oauth-2-1-vs-2-0/>
6. OpenID Connect vs OAuth2: The Differences and How to Choose - SuperTokens, дата последнего обращения: декабря 6, 2025, <https://supertokens.com/blog/openid-connect-vs-oauth2>
7. OAuth 2.0 and OpenID Connect for API Security: A Technical Deep Dive | by Okan Yıldız, дата последнего обращения: декабря 6, 2025, <https://medium.com/@okanyildiz1994/oauth-2-0-and-openid-connect-for-api-security-a-technical-deep-dive-ab371ab3ae96>
8. Security definition uses the deprecated implicit flow on OAuth2, дата последнего обращения: декабря 6, 2025, <https://docs.prismacloud.io/en/enterprise-edition/policy-reference/api-policies/openapi-policies/bc-openapi-12>
9. Implicit flow vs. Authorization code flow: Why implicit flow is dead? - Logto blog, дата последнего обращения: декабря 6, 2025, <https://blog.logto.io/implicit-flow-is-dead>
10. What's Wrong With the OAuth2 Implicit Grant? - FusionAuth, дата последнего обращения: декабря 6, 2025, <https://fusionauth.io/blog/whats-wrong-with-implicit-grant>
11. OAuth 2.0 vs 2.1: What's Changed and Why It Matters for Developers - Aembit, дата последнего обращения: декабря 6, 2025, <https://aembit.io/blog/oauth-2-1-guide-migration-security/>
12. Defending OAuth: Common attacks and how to prevent them - WorkOS, дата последнего обращения: декабря 6, 2025, <https://workos.com/blog/oauth-common-attacks-and-how-to-prevent-them>
13. OAuth2 Implicit Grant and SPA - Auth0, дата последнего обращения: декабря 6, 2025, <https://auth0.com/blog/oauth2-implicit-grant-and-spa/>
14. Why is there an "Authorization Code" flow in OAuth2 when "Implicit" flow works so well?, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/13387698/why-is-there-an-authorization-code-flow-in-oauth2-when-implicit-flow-works-s>
15. Implement Authorization Code flow with PKCE - Implement authorization by grant type | Okta Developer, дата последнего обращения: декабря 6, 2025, <https://developer.okta.com/docs/guides/implement-grant-type/authcodepkce/main/>
16. Proof Key of Code Exchange (PKCE) - SecureAuth Product Docs, дата последнего обращения: декабря 6, 2025, <https://docs.secureauth.com/ciam/en/proof-key-of-code-exchange--pkce-.html>
17. OAuth 2.0: Implicit Flow is Dead, Try PKCE Instead | Postman Blog, дата последнего обращения: декабря 6, 2025, <https://blog.postman.com/pkce-oauth-how-to/>
18. OAuth 2.0 Security Best Practices: From Authorization Code to PKCE | by Başak Erdoğan, дата последнего обращения: декабря 6, 2025, <https://medium.com/@basakerdogan/oauth-2-0-security-best-practices-from-authorization-code-to-pkce-beccdbe7ec35>
19. Proof Key for Code Exchange (RFC 7636) - Authlete, дата последнего обращения: декабря 6, 2025, <https://www.authlete.com/developers/pkce/>
20. RFC 7636 - Proof Key for Code Exchange by OAuth Public Clients - IETF Datatracker, дата последнего обращения: декабря 6, 2025, <https://datatracker.ietf.org/doc/html/rfc7636>
21. Generate the Code Challenge | Authentication (OAuth) - Autodesk Platform Services, дата последнего обращения: декабря 6, 2025, <https://aps.autodesk.com/en/docs/oauth/v2/tutorials/code-challenge>
22. Authorization Code Flow with Proof Key for Code Exchange (PKCE) - Auth0, дата последнего обращения: декабря 6, 2025, <https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow-with-pkce>
23. What is JWKS? JSON Web Key Set Explained with Examples - Authgear, дата последнего обращения: декабря 6, 2025, <https://www.authgear.com/post/what-is-jwks>
24. How to properly consume OpenID Connect jwks\_uri metadata in C#? - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/47121732/how-to-properly-consume-openid-connect-jwks-uri-metadata-in-c>
25. JSON Web Token Claims - Auth0, дата последнего обращения: декабря 6, 2025, <https://auth0.com/docs/secure/tokens/json-web-tokens/json-web-token-claims>
26. How OpenID Connect Works - OpenID Foundation, дата последнего обращения: декабря 6, 2025, <https://openid.net/developers/how-connect-works/>
27. ID token claims reference - Microsoft identity platform, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/entra/identity-platform/id-token-claims-reference>
28. ID Token Structure - Auth0, дата последнего обращения: декабря 6, 2025, <https://auth0.com/docs/secure/tokens/id-tokens/id-token-structure>
29. JWT Claims in OpenID Connect: Explained (With Examples) - Criipto, дата последнего обращения: декабря 6, 2025, <https://idura.eu/blog/jwt-claims-example>
30. An Illustrated Guide to OAuth and OpenID Connect - YouTube, дата последнего обращения: декабря 6, 2025, <https://www.youtube.com/watch?v=t18YB3xDfXI>
31. Token formats - OpenIddict, дата последнего обращения: декабря 6, 2025, <https://documentation.openiddict.com/configuration/token-formats>
32. Understanding IdentityServer4's Removal and Its Alternatives - FeatBit, дата последнего обращения: декабря 6, 2025, <https://www.featbit.co/articles2025/identityserver4-removed-github-alternatives>
33. Identity management solutions for .NET web apps | Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/aspnet/core/security/identity-management-solutions?view=aspnetcore-10.0>
34. What is Wrong with IdentityServer4 and What Alternatives to Choose? – Legacy Software Modernization Company | ModLogix, дата последнего обращения: декабря 6, 2025, <https://modlogix.com/blog/identityserver4-alternatives-best-options-and-the-near-future-of-identityserver/>
35. IdentityServer4 vs AspNet.Security.OpenIdConnect.Server vs OpenIddict - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/46608920/identityserver4-vs-aspnet-security-openidconnect-server-vs-openiddict>
36. ASP.NET Core integration - OpenIddict, дата последнего обращения: декабря 6, 2025, <https://documentation.openiddict.com/integrations/aspnet-core>
37. Implementing token validation in your APIs - OpenIddict, дата последнего обращения: декабря 6, 2025, <https://documentation.openiddict.com/guides/getting-started/implementing-token-validation-in-your-apis>
38. Encryption and signing credentials - OpenIddict, дата последнего обращения: декабря 6, 2025, <https://documentation.openiddict.com/configuration/encryption-and-signing-credentials>
39. Authorization Server with OpenIddict: Authorization Code Flow - Lê Gimenes - Medium, дата последнего обращения: декабря 6, 2025, <https://legimenes.medium.com/authorization-server-with-openiddict-authorization-code-flow-e96669864727>
40. Unity WebGL + PlayFab Authorization in 20 Minutes | Okta Developer, дата последнего обращения: декабря 6, 2025, <https://developer.okta.com/blog/2021/02/26/unity-webgl-playfab-authorization>
41. Deep linking - Unity - Manual, дата последнего обращения: декабря 6, 2025, <https://docs.unity3d.com/6000.2/Documentation/Manual/deep-linking.html>
42. How to redirect back to unity mobile app from oauth sign in? - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/60762393/how-to-redirect-back-to-unity-mobile-app-from-oauth-sign-in>
43. Configure JWT bearer authentication in ASP.NET Core | Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/configure-jwt-bearer-authentication?view=aspnetcore-10.0>
44. OpenID Connect Discovery | Swagger Docs, дата последнего обращения: декабря 6, 2025, <https://swagger.io/docs/specification/v3_0/authentication/openid-connect-discovery/>
45. OpenID Connect | Sign in with Google, дата последнего обращения: декабря 6, 2025, <https://developers.google.com/identity/openid-connect/openid-connect>
46. Implementing OpenID Connect with OpenIddict - DEV Community, дата последнего обращения: декабря 6, 2025, <https://dev.to/naeemsahil/implementing-openid-connect-with-openiddict-4fmp>
47. Claim destinations server - OpenIddict, дата последнего обращения: декабря 6, 2025, <https://documentation.openiddict.com/configuration/claim-destinations>