Аутентификация и авторизация. oAuth, JWT, forms. Сессии. Ролевое разграничение доступа к ресурсам

Обзор подходов к управлению доступом и сессиями для разработчиков

Выполнил Каспранов Камиль

Аутентификация

Это процесс проверки личности пользователя — подтверждение того, что человек действительно тот, за кого себя выдает.

Пример:

1. Ввод логина и пароля
2. Отпечаток пальца или Face ID
3. SmS-код или одноразовый код из приложения (2FA)

Авторизация

Это процесс определения прав доступа уже аутентифицированного пользователя — то есть какие действия он может выполнять или к каким ресурсам имеет доступ.

Пример:

1. Пользователь может читать файлы, но не может их удалять
2. Администратор имеет доступ к настройкам системы, а обычный пользователь — нет

Важно:

- Сначала происходит аутентификация, потом — авторизация.
- Без успешной аутентификации авторизация невозможна.

Карта Современных Подходов

Существует четыре основных парадигмы аутентификации в веб-приложениях, каждая со своими компромиссами.

Form-based Auth

Session-based Auth

OAuth / OIDC

JWT (Stateless)

1. Аутентификация на Основе Форм

Простота и Классика

Механизм

Пользователь вводит учетные данные в форме. Сервер проверяет их, часто используя хеши паролей.

Преимущества

Высокая простота внедрения

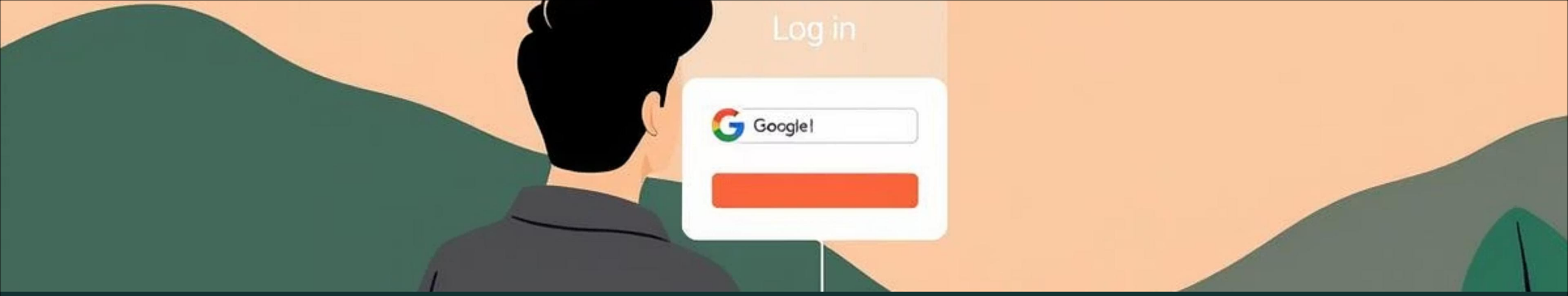
- Привычен для пользователей
- Легко реализовать даже начинающему разработчику.
- Независимость от внешних сервисов

```
document.getElementById('loginForm').addEventListener('submit', async (e) => {
    e.preventDefault();
    const username = document.getElementById('username').value;
    const password = document.getElementById('password').value;

    try {
      const response = await fetch('/api/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, password })
      });

      const data = await response.json();

      if (response.ok) {
        window.location.href = '/dashboard';
      } else {
        alert( 'Ошибка входа');
      }
    } catch (err) {
      alert('Нет связи с сервером');
    }
  });
```



2. OAuth 2.0 и OpenID Connect (OIDC)

Делегированная Аутентификация

OAuth 2.0

Стандарт **авторизации** (предоставления доступа). Позволяет приложению A получить доступ к ресурсам пользователя в приложении B.

Пример: дать приложению доступ к вашему Google Drive.

OpenID Connect

Уровень **аутентификации** поверх OAuth 2.0. Он подтверждает личность пользователя, возвращая ID Token.

Пример: войти в ваше приложение через аккаунт Google.

3. Сессионная Аутентификация

Управление Состоянием (Stateful)

Создание

Сервер создает уникальный ID сессии после успешного входа.

Хранение

ID сессии сохраняется в зашифрованном `HTTP-only` cookie.

Плюсы

- Надежный контроль: мгновенное аннулирование сессии (logout)
- Простота управления правами

Минусы

- Проблема масштабирования (Stateful): сервер должен хранить все сессии
- Требует выделенного хранилища сессий

4. JWT: JSON Web Token

Stateless подход для современных архитектур

```
const { SignJWT, jwtVerify } = require('jose');
const secret = new TextEncoder().encode('my-super-secret-key-must-be-32-bytes-long!');

const payload = {
  userId: 42,
  role: 'user',
  email: 'user@example.com',
  iat: Math.floor(Date.now() / 1000),
  exp: Math.floor(Date.now() / 1000) + 3600
};

async function signJWT() {
  const jwt = await new SignJWT(payload)
    .setProtectedHeader({ alg: 'HS256' })
    .setIssuedAt()
    .setExpirationTime('1h')
    .sign(secret);

  console.log('JWT:', jwt);
  return jwt;
}
```

Stateless

Сервер не хранит сессий. Информация о пользователе (Payload) зашифрована в самом токене.

Подпись

Токен подписан секретным ключом, что гарантирует его целостность и аутентичность.

Передача

Клиент отправляет токен в заголовке

Пример:
Header - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
Payload -eyJ1c2VySWQiOjQyLCJyb2xlIjoidXNlciIsImVtYWlsIjoidXNlckBleGFtcGxlLmNvbSIsImlhdCI6MTc2MTYxMjAwMCwiZXhwIjoxNzYxNjE1NjAwfQ.
Signature - 7Z8pFkqWdDv7bHs7gKo8n3hT3iP7eCz8r7qK8u3YwXc

Преимущества и Недостатки JWT

JWT идеально подходит для микросервисов, но имеет свои проблемы безопасности.

Высокая Масштабируемость

Любой сервер может проверить токен, не обращаясь к общему хранилищу.

Идеально для Микросервисов

Простота распределенной аутентификации между сервисами.

Проблема Отзыва

Токен нельзя отозвать до истечения срока действия (TTL), что требует короткого срока жизни токенов.

Риск XSS/CSRF

Требует внимательного хранения (лучше в HTTP-only cookies, чем в LocalStorage).

Сравнение: Stateful vs. Stateless

Выбор подхода влияет на архитектуру системы и стратегию масштабирования.

Параметр	Session-based (Stateful)	JWT (Stateless)
Хранение	Сервер (DB/Кэш)	Клиент (Токен)
Масштабирование	Сложности из-за необходимости синхронизации состояния	Высокая простота
Отзыв	Мгновенный	Сложный
Микросервисы	Требуется единый Session Store	Встроенная поддержка

Ролевое разграничение доступа к ресурсам

(Role-Based Access Control, RBAC) — это метод управления правами

пользователей в системе на основе ролей,

а не отдельных учётных записей.

Преимущество	**Объясниние**
Простота управления	Чтобы дать новые права — меняете роль, а не сотни пользователей.
Аудит и прозрачность	Легко понять, кто что может: «Админ видит всё».
Снижение ошибок	Меньше ручной настройки → меньше человеческих ошибок.
Соответствие стандартам	Требуется в регулируемых отраслях (финансы, медицина).

Спасибо за внимание