

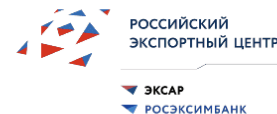
Проблемы аутентификации в современных frontend приложениях



Кто я

1. 10+ лет в вебе. Прошел путь fullstack, teamlead.
2. Пишу на PHP (Laravel/Bitrix), Typescript, Nodejs
3. Влюблен в Linux, OpenSource, React.
4. Работаю СТО в Вебпрактик (*отдел разработки 50+ человек*)
5. Один из организаторов Ростовского PHP сообщества

В Вебпрактик мы пишем сервисы для корпораций





Что побудило сделать доклад

1. Я не нашел единого источника информации который бы охватил все аспекты построения безопасной авторизации на JWT токенах для SPA приложений
2. Я не нашел библиотеки которая всецело бы удовлетворяла потребности фронтендера и инкапсулировала всю сложность



Вопросы в зал

- Кто из вас реализовывал аутентификацию в SPA приложениях?
- Кто делал stateless авторизацию на JWT токенах?
- Кто читал OWASP рекомендации (или RFC 8725 JSON Web Token Best Current Practices) как сделать это безопасно?



Как мы пришли к вопросу

- Мы создаем приложений и сайты для корпораций
- В 18 году у нас стало больше MSA/SOA/Гибрид решений и SPA приложений
- Начали переосмысливать подходы к авторизации, начали делать первые приложения на JWT токенах



Немного теории



- **Идентификация** — процесс распознавания пользователя по его идентификатору.
- **Аутентификация** — процедура проверки подлинности, доказательство, что пользователь именно тот, за кого себя выдает.
- **Авторизация** — предоставление определённых прав.

Терминология



Основные современные виды аутентификации в веб приложениях

С технической точки зрения.

1. Session
2. JWT
3. OAuth 2.0 / OIDC
4. AD/LDAP федерации (в т.ч. NTLM/Kerberos)
5. Basic Http



Основные современные виды аутентификации в веб приложениях

С технической точки зрения.

1. Session
2. JWT (custom)
3. OAuth 2.0 / OIDC
4. AD/LDAP федерации (в т.ч. NTLM/Kerberos)
5. Basic Http



Почему не рассматриваем OIDC/OAuth?



OIDC/OAuth + Gatekeeper = ❤️



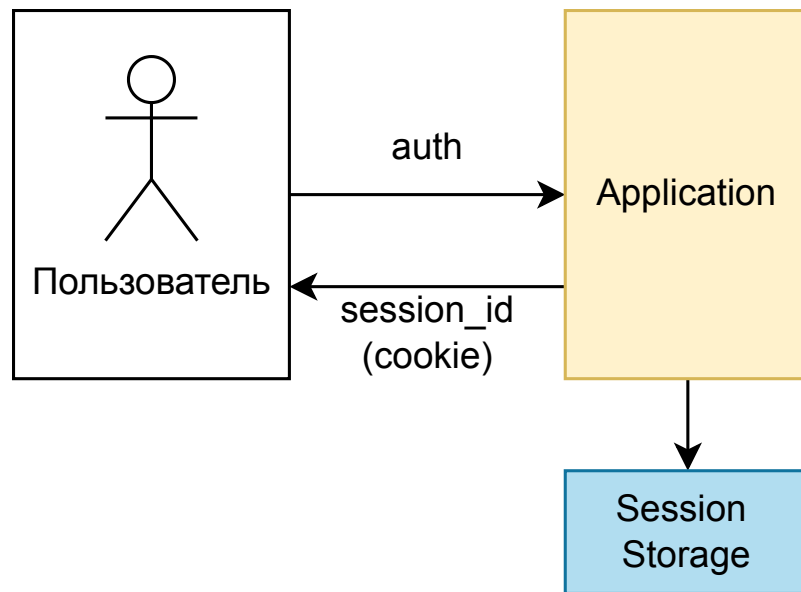
Stateless (JWT) vs Stateful (session) авторизация



Stateful (session) авторизация

Классическая аутентификация в монолитных приложениях

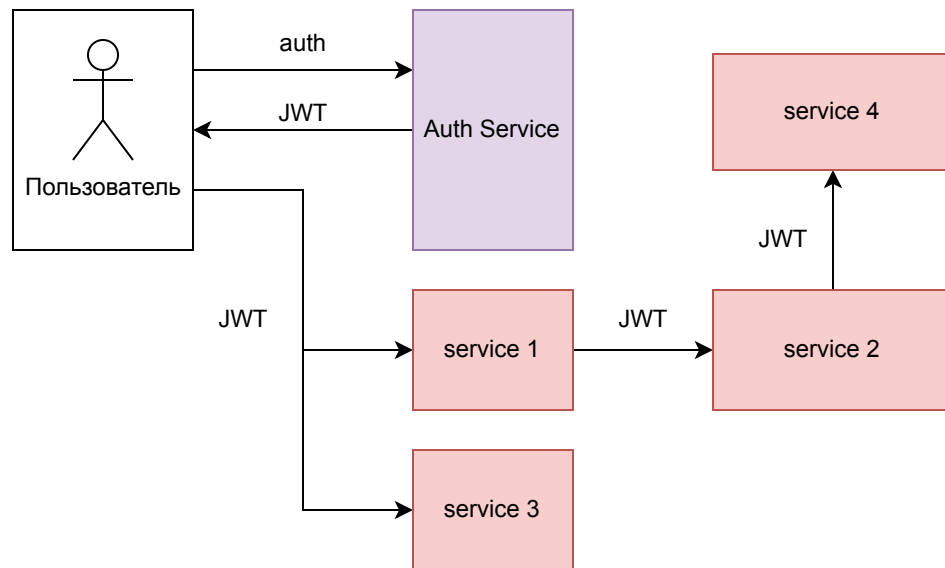
- + Простота
- + Проще обеспечить безопасность
- Скорость
- Масштабирование
- Плохо подходит SOA/MSA





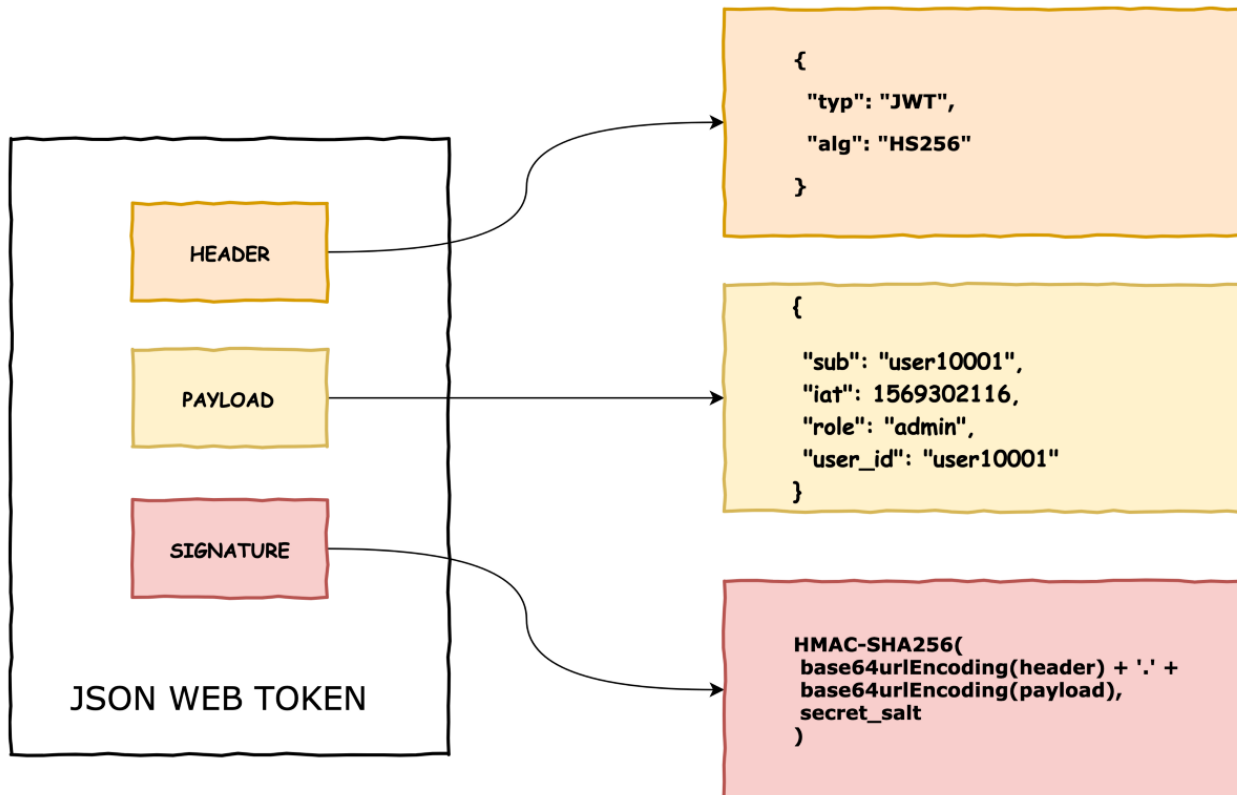
Stateless (JWT)

- + Масштабирование
- + Скорость
- Сложнее обеспечить безопасность





JWT



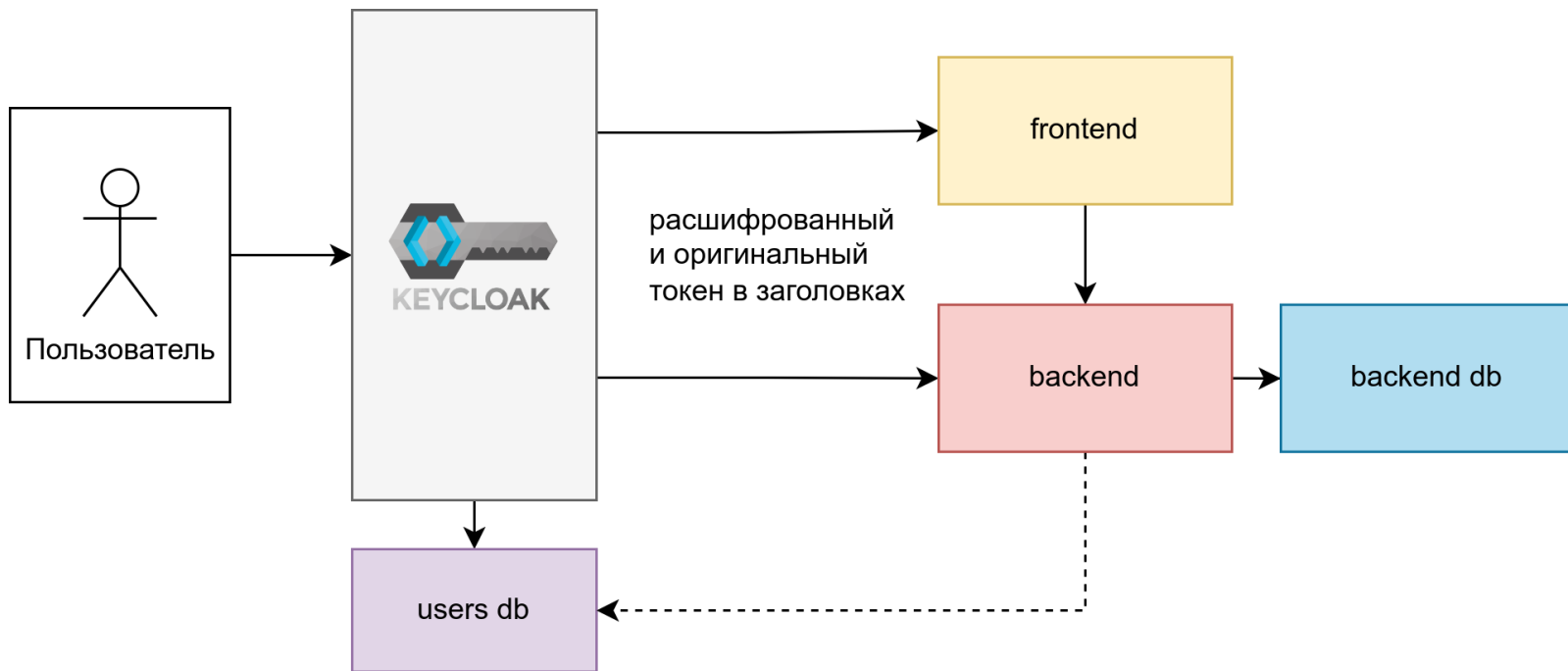


Где разместить аутентификацию?

1. Корпоративный SSO на базе Keycloak или OryStack.
2. Аутентификация на стороне бекенда
3. Аутентификация и авторизация на фронтенд-сервисе

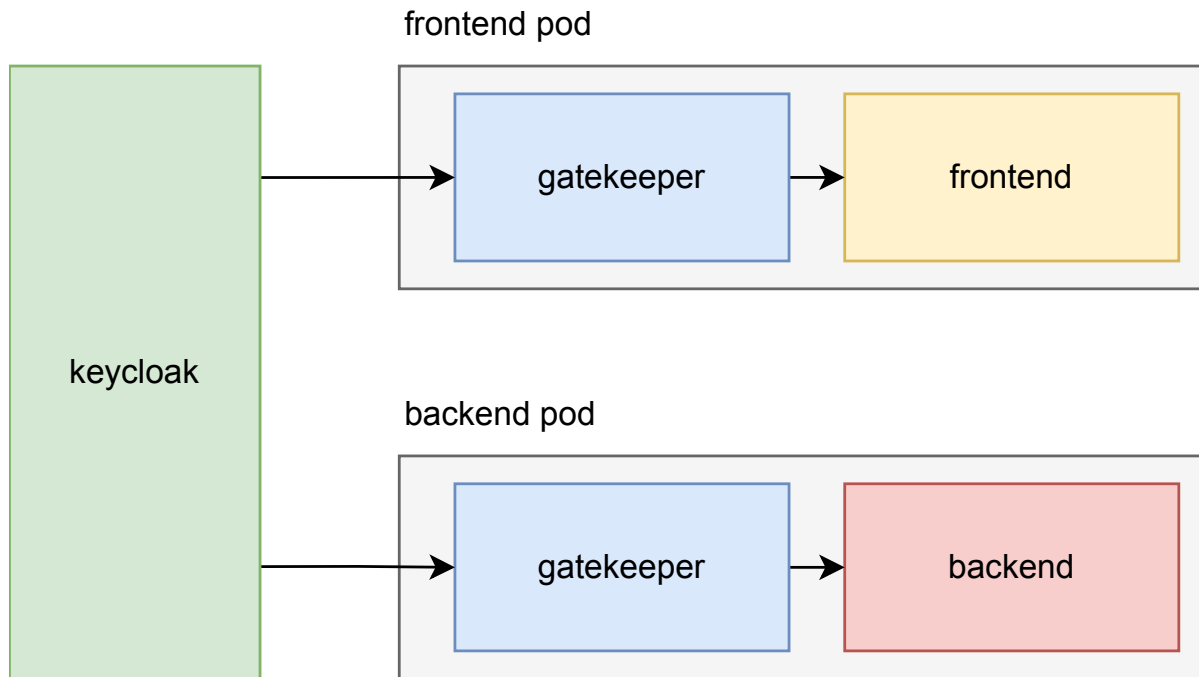


Подходы: корпоративный SSO Keycloak/Ory





Подходы: корпоративный SSO Keycloak/Ory



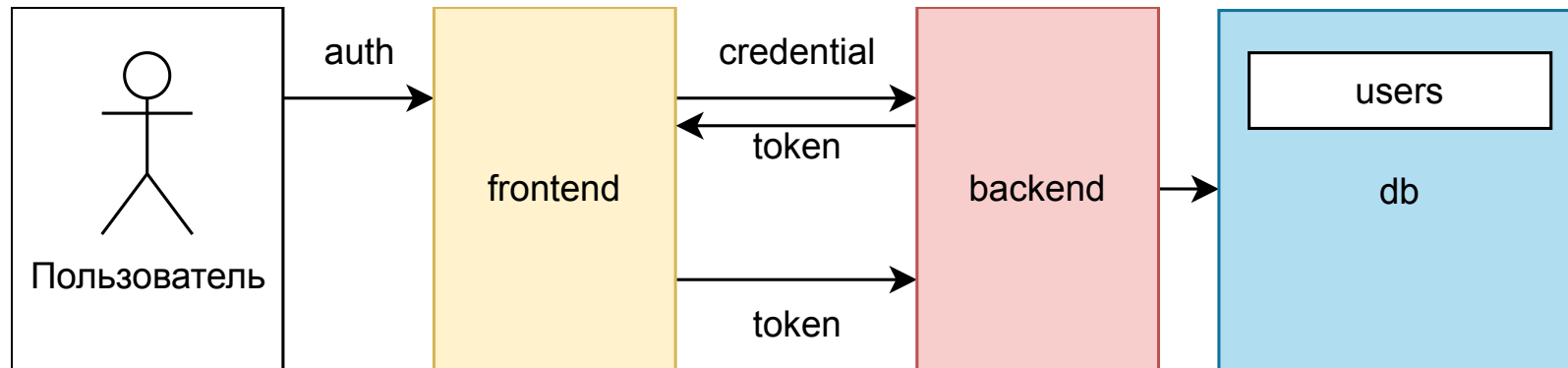


Подходы: корпоративный SSO Keycloak/Ory

- + Забирает почти всю боль с приложений бекенда и фронтенда
- + Рекомендации по безопасности из коробки
- + Нет дублирования кода авторизации в каждом микросервисе
- + Stateless, JWT
- Доп узел, который нужно мониторить, поддерживать, обновлять, обслуживать
- Может возникнуть кейс с необходимостью управления разделяемой (общей) базой между сервисами



Подходы: аутентификация на бекенде



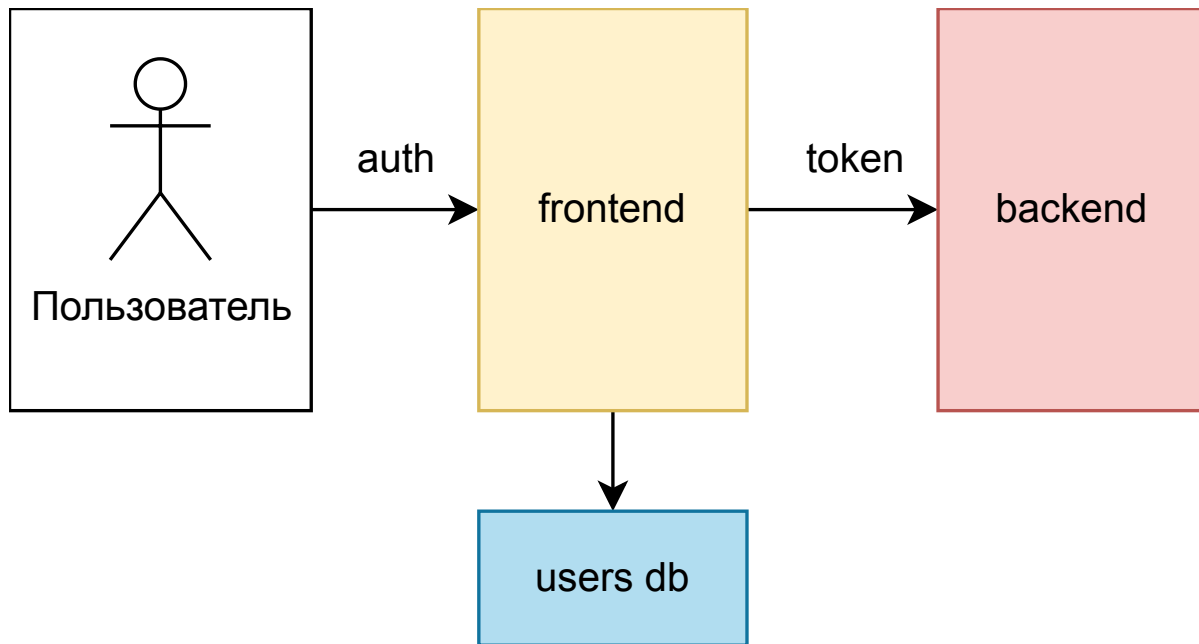


Подходы: аутентификация на бекенде

- + За хранение и обработку пользователей отвечает 1 сервис
- + Stateless JWT, хотя можно и statefull бахнуть
- Все писать самому. На бекенде есть готовые решения, на фронте нужно будет писать.



Подходы: аутентификация и хранение пользователей на фронтенд-сервисе





Подходы: аутентификация и хранение пользователей на фронтенд-сервисе

- + За хранение и обработку пользователей отвечает 1 сервис
- За хранение и обработку пользователей отвечает 1 сервис
- Подойдет ограниченному количеству сервисов

Next auth предлагает такую схему из коробки как один из вариантов.



Разберем очень частый случай для типового SPA приложения

- Аутентификация на бекенде
- Бекенд отдает REST API + JWT токен



Какие вопросы перед нами встают?



Где хранить JWT токен?

1. В local storage?

Нежелательно. Если многодоменная конфигурация то можно по рекомендациям OWASP использовать `SessionStorage` + `fingerprint`.

2. В cookie `HttpOnly`



Как читать данные с токена?

1. Локальное хранение

- Просто читаем данные с токена который хранится? **плохо**
- А если токен зашифрован? (Рекомендация OWASP)

2. cookie http_only:

=> делаем запрос на сервер при загрузке страницы (best practices)



Сколько должен жить JWT токен?

1. best practices 5-15 минут



А как его обновлять?

1. Refresh token



Где хранить Refresh token?

1. cookie http_only / local storage



Сколько хранить Refresh token?

1. От нескольких часов, до нескольких недель



Механика обновления access token

1. cookie `http_only`: происходит фоном на бекенде
2. локальное хранение: `response interceptor` клиента на 401, с перезапросом



Механика обновления access token

```
import { ErrorHandler, Injectable } from "@angular/core"
import { HttpResponse, HttpEvent, HttpHandler, HttpInterceptor, HttpRequest } from "@angular/common/http"

import { catchError, switchMap, take, tap } from "rxjs/operators"
import { BrowserStorageService } from "../shared-services/browser-storage.service"
import { BrowserStorageVariables } from "../model"
import { Observable, of, Subject, throwError } from "rxjs"
import { Store } from "ngrx/store"
import { RootState } from "../../web-navigator/app/core/model"
import { authPath } from "../constants"
import { AuthService } from "../../web-navigator/app/auth/services/auth.service"
import * as fromActions from "../../web-navigator/app/core/actions"

@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  constructor(private browserStorage: BrowserStorageService,
    private errorHandler: ErrorHandler,
    private store$: Store,
    private authService: AuthService) {}

  /**
   * Флаг говорящий о том что запрос на получение нового access токена в процессе
   */
  private isGetAccessTokenGetInProgress: boolean = false

  /**
   * Уведомитель, который сообщает что получен новый access token
   */
  private accessToken$: Subject = new Subject()

  public intercept(req: HttpRequest, next: HttpHandler): Observable {
    return this.getRequest(req, next).pipe(
      catchError((error: HttpResponse) => {
        if (error.url === null) {
          return throwError(error)
        }

        if (error.url.includes(authPath)) {
          this.isGetAccessTokenGetInProgress = false
          this.store$.dispatch(fromActions.destroyApp())
          return of(null)
        }
      })
    )

    /**
     * Если код ошибки не 401, значить что-то не так,
     * никак не связанное с авторизацией.
     * Просто прокидываем ошибку дальше
     */
    if (error.status !== 401) {
      this.errorHandler.handleError(error)
      return throwError(error)
    }

    /**
     * Если получение нового access в процессе,
     * то кидаем пришедший запрос в очередь ожидания
     */
    if (this.isGetAccessTokenGetInProgress) {
      return this.accessToken$.pipe(
        take(1),
        switchMap(() => this.getRequest(req, next))
      )
    }

    /**
     * Если код добрался сюда, пройдя все проверки,
     * то это означает что именно на этом запросе сгорел access token,
     * так что мы говорим что получение в процессе и идем на сервер
     * авторизации за новым
     */
    this.isGetAccessTokenGetInProgress = true

    /**
     * Достаем refresh token
     */
    const refreshToken: string | null = this.browserStorage.onGet(BrowserStorageVariables.refreshToken)
    const orgKey: string | null = this.browserStorage.onGet(BrowserStorageVariables.orgKey)

    if (refreshToken === null || orgKey === null) {
      return throwError(new Error("Tokens not defined"))
    }

    return this.authService.authorizeByRefresh(refreshToken, orgKey).pipe(
      tap(() => {
        /**
         * Сохраняем новые refresh и access токены
         */
        this.browserStorage.onSet(BrowserStorageVariables.accessToken, v.access_token)

        /**
         * Говорим очереди всех запросов, что теперь можно
         * выполняться
         */
        this.accessToken$.next()
        this.isGetAccessTokenGetInProgress = false
      })
    )

    switchMap(() => {
      /**
       * Выполняем, тот самый запрос на котором
       * мы впервые узнали что token сгорел
       */
      return this.getRequest(req, next)
    })
  )
}

private getRequest(req: HttpRequest, next: HttpHandler): Observable {
  const accessToken: string | null = this.browserStorage.onGet(BrowserStorageVariables.accessToken)

  if (accessToken === null) {
    return next.handle(req)
  }

  return next.handle(req.clone({headers: req.headers.set("Authorization", `Bearer ${accessToken}`)}))
}
```



Ротация Refresh токенов

- При запросе нового access token, выдавать в т.ч. новый refresh token, делая старый устаревшим
- Идем дальше: объединять цепочку токенов в семейство. При попытке использовании старого refresh token - убивать всю семью.



Как делать запросы на сервисы?

1. cookie http_only: ничего делать не нужно
2. локальное хранение: request interceptor с Authorization header



Как отозвать токен?

- blacklist (OWASP)



blacklist? Но как же stateless?



Как защитить токен от кражи? (Token Sidejacking)

- csrf token в виде отдельной http_only cookie в котором хранить user context + random SHA256



**Хочу не изобретать велосипеды
(каждый раз), а взять готовое/
системное решение.**



Таблица готовых решений

Библиотека	<u>react-token-auth</u>	<u>react-auth-kit</u>	<u>oidc-react</u>	<u>react-oidc</u>	<u>useauth.dev</u>	NextAuth
Год выхода	2020	2020	2020	2020	2020	2020
Stars	79	192	204	364	2600	11600
Последний релиз	2021	2022	2022	2022	2020	2022
Комментарий	Автор рекомендует использовать библиотеку в крайнем случае и отказывается от ответственности =)				Всего 2 релиза и не развивается	



**Берем самое популярное решение:
NextAuth**



Разбор NextAuth: Безопасность

- + реализовано шифрование JWS / JWE / JWK
- + httpOnly SameSite Secure cookie
- + запрос данных расшифрованной cookie с сервера
- + Синхронизация вкладок, автопроверка, поддержка активности
- + CSRF
- + Ротация токенов
- Нет Refresh токена => нет механизма отзыва токенов



Разбор NextAuth: Функциональность


1. Готовые обвязки с 50+ разными провайдерами авторизации (facebook, google, discord, github, yandex, vk, azure, atlassian, OAuth, OIDC и мн другие)
2. Провайдер авторизации через email
3. Готовые удобные хуки
4. SSR Friendly
5. Провайдеры к базам данных: typeorm, prisma, sequelize и пр



Главная ложка дегтя

1. Основной акцент Full Stack + Built for Serverless
2. Провайдер авторизации через Credentials есть, но JWT токены выпускает сам NextAuth



A man with dark hair and a mustache is shown from the waist up. He is wearing a brown blazer over a blue and white vertically striped button-down shirt. He is standing in front of large, light-colored concrete pillars. In the background, a utility pole with cross-arms is visible against a pale sky. The overall color palette is somewhat muted, with a slight purple/pink tint.

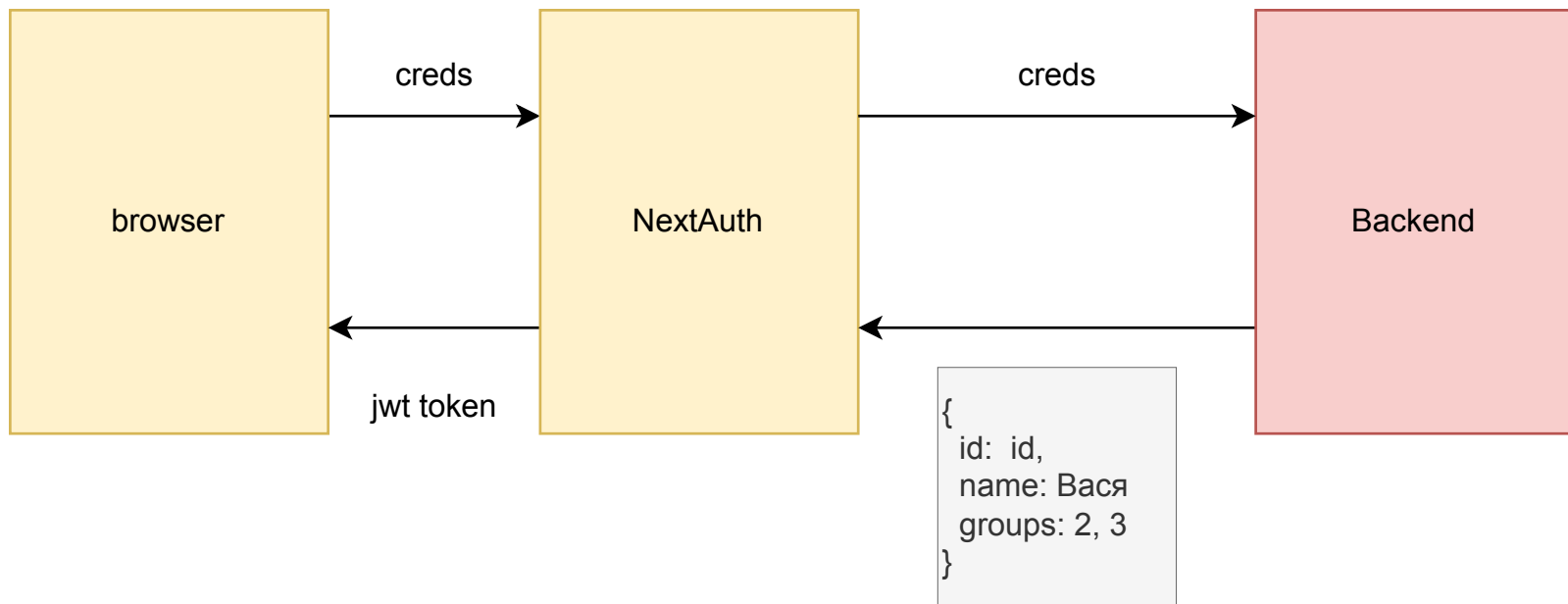
Ну да, ну да. Пошёл я на хер.



Как это может работать



Авторизация с custom credential provider

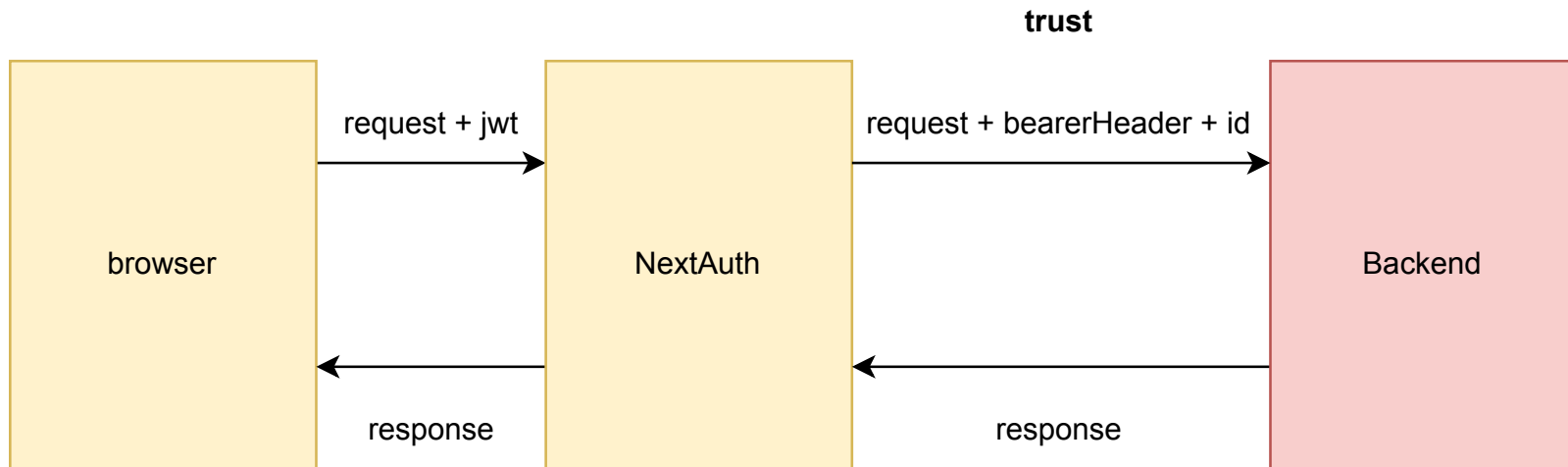




А как делать запросы к бекенду?



BFF proxy + trust

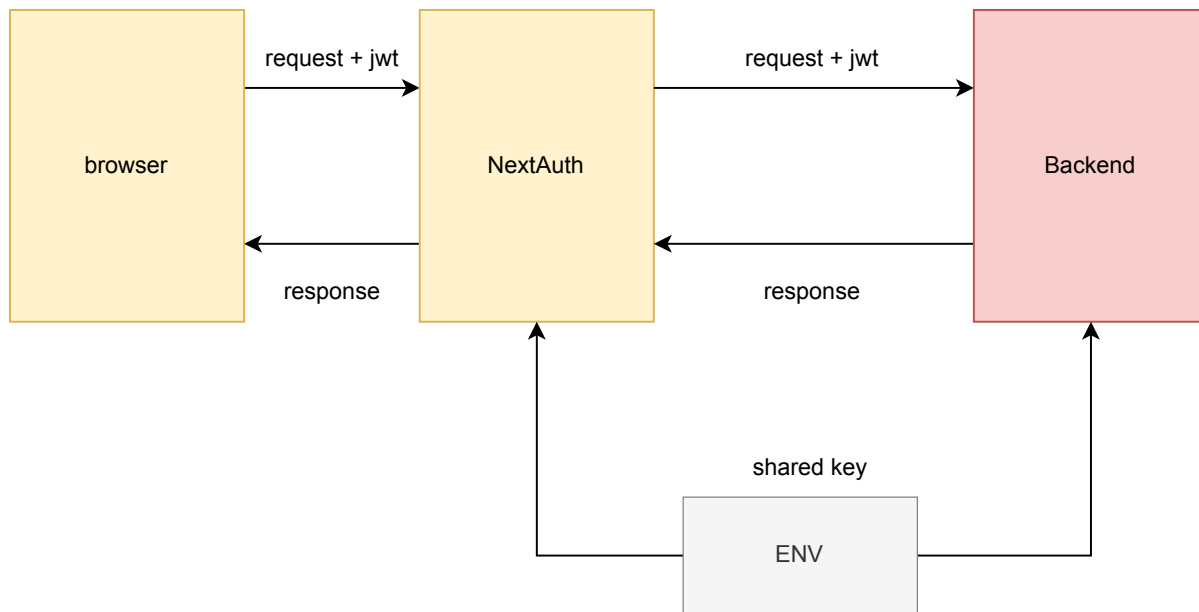


* Закрываем бек от внешки

➖ Необходимо безоговорочное доверие сервису фронта



Shared jwt encrypted key



❌ Специфичный алгоритм шифрования **из коробки**



Nextauth: выводы

Все неплохо с безопасностью

Отлично подходит когда serverless, или fullstack проектов

Отлично зайдет когда бекенд предоставляет oAuth авторизацию

В случае со стандартной авторизацией с бекендом - подойдет, но придется повозиться. И договорится с беком о правилах игры.



Таблица готовых решений

Библиотека	<u>react-token-auth</u>	<u>react-auth-kit</u>	<u>oidc-react</u>	<u>react-oidc</u>	<u>useauth.dev</u>	NextAuth
Год выхода	2020	2020	2020	2020	2020	2020
Stars	79	192	204	364	2600	11600
Последний релиз	2021	2022	2022	2022	2020	2022
Комментарий	Автор рекомендует использовать библиотеку в крайнем случае и отказывается от ответственности =)				Всего 2 релиза и не развивается	



Обзор: useAuth

- ➖ Не обновлялся 2 года
- ➖ Слабая проработка безопасности, токены хранятся локально
- ➖ Все равно писать интерцепторы
- ➕ Неплохая документация



Обзор: react-auth-kit

- + Регулярно обновляется
- Мало звезд и скачиваний
- Слабая проработка безопасности, токены хранятся локально



Выводы

- На фронтенде нет хороших универсальных готовых решений закрывающих все основные требования безопасности
- Ближе всех к решению вопроса Nextjs, но он несет дополнительные сложности
- Идеальный вариант - oids аутентификация
- Если беретесь за самостоятельную реализацию - погрузитесь в вопрос или хотя бы сверьтесь с чеклистом ниже



Полезные ссылки

- [OWASP JWT Cheat Sheet](#)
- [{JWT}.{Attack}.Playbook](#)
- [JSON Web Token Best Current Practices](#)
- [NextAuth](#)
- [Таблица готовых решений](#)

Спасибо! 🙄

Ваши вопросы

Telegram: [@northleshiy](https://www.t.me/northleshiy)

