Security Scheme MyShop App

1. OAuth2 Overview

In short, OAuth2 is a more versatile and simple approach compared to classical HTTP-sessions.

As a result, it can be employed by various types of applications: web, mobile, desktop.

Pros:

- User management and all validations are removed from the backend application and are shifted to a dedicated authorization server Keycloak.
- More universal tokens are utilized, which are included in requests and will function regardless of the domain or application type.
- There are fewer connections and dependencies between microservice nodes throughout the entire system. Scaling the backend becomes easier since there is no dependency on browser sessions.

2. General steps, made for user authentication: PKCE

4.1. Authorization Flow for Native Apps Using the Browser

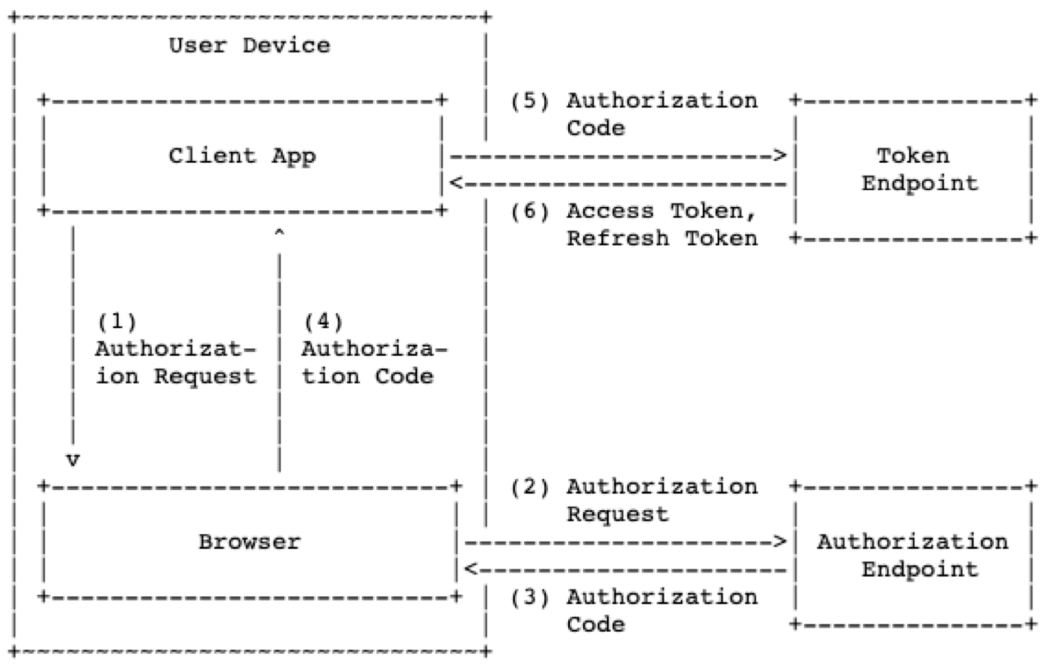


Figure 1: Native App Authorization via an External User-Agent

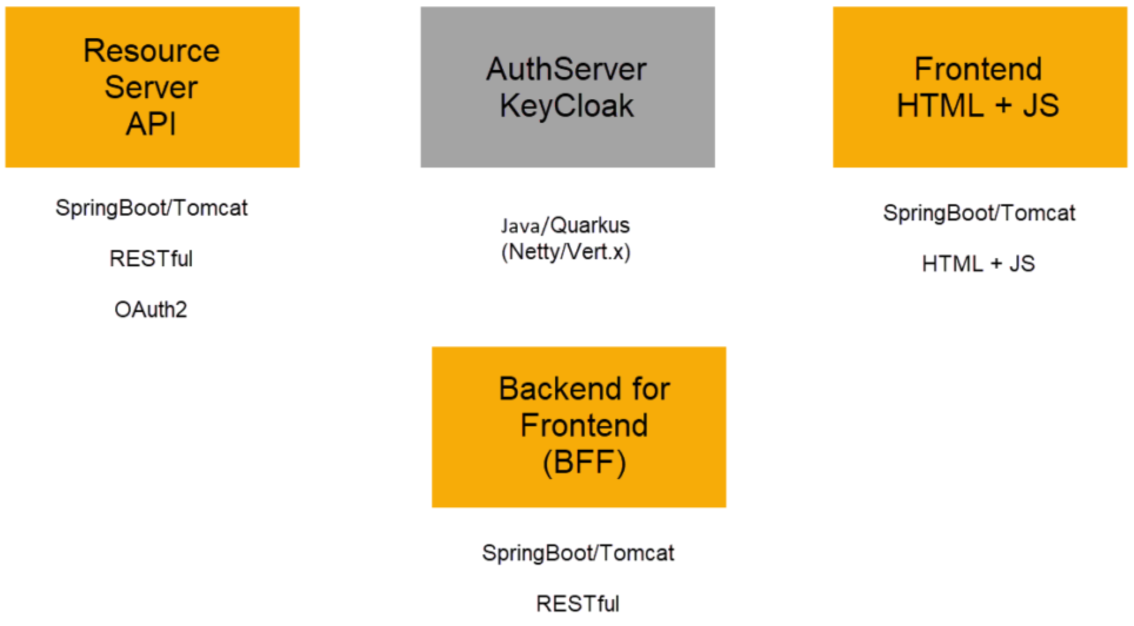
Main steps which are made for Proof Key for Code Exchange (PKCE) in OAuth2 Authorization flow:

1. Frontend (User) requests for authorization directly from Keycloak Authorization Server (later Keycloak). Enters login and password
2. After successful authorization get Authorisation code from Keycloak and send them to BFF. Note: Authorisation code is not yet Access Token
3. Bff requests Access tokens from Keycloak using Authorisation code
4. Bff receives Access Tokens and requests data from Backend using tokens

3. BFF Pattern

App uses a Keycloak third party authorization server. It implements OAuth2 security protocols, which are nowadays modern security standards.

Let’s look on overall schema of security architecture



In brief, BFF (Backend for Frontend) enables the creation of more secure server-side cookies and also stores tokens in the browser in a safer manner (inaccessible from JavaScript). Only the BFF can access these cookies.

All client requests first pass through the BFF before being forwarded to the Resource Server. It's often used not only to safeguard the frontend but also to organize a convenient API.

The trendiest solution for token protection currently is the BFF pattern - Backend for Frontend.

Its motto: "No tokens in the browser Policy" - "say no to token storage in the browser."

However, it's more accurate to state: storing tokens in more secure server-side cookies.

Yet, it doesn't provide 100% protection against XSS-attacks since malicious code can still execute (the server might not differentiate it in most cases). But at least the load is removed from the frontend.

BFF's strength lies in its ability to analyze and control suspicious requests where hackers can't reach. For example, if a particular operation is executed multiple times, the BFF can revoke all tokens for suspicious actions.

Cookies will never enter the browser storage (localStorage), thereby preventing hackers from accessing them.

All tokens will automatically be sent with requests to the BFF by the browser through secure cookies.

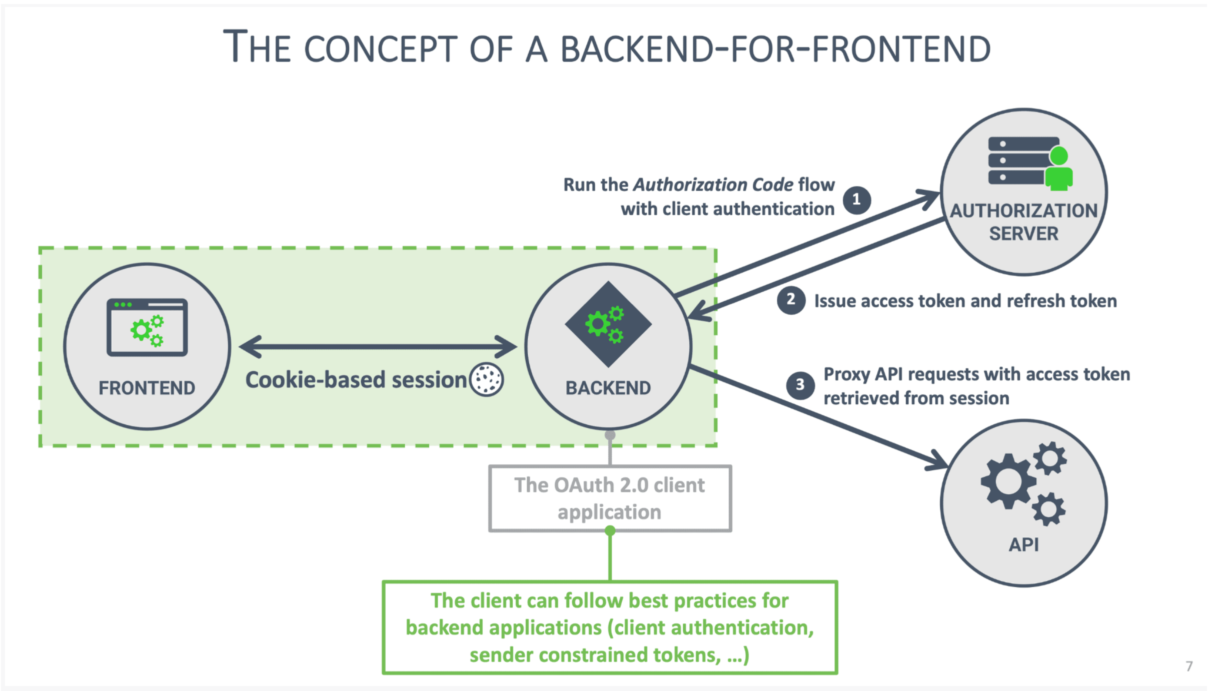
1. Backend check for Access tokens validity. It requests Keycloak for it. If successful, Backend sends data back to BFF. Bff proxies data to Frontend

Key moment: Frontend by itself doesn’t receive any tokens and doesn't do any operations with them. Tokens are still stored on frontend but in secured server side cookies and they are not accessible from frontend javascript

Externally, nothing has changed for the user in fact – he enters a username and password and logs in, receives his data.

4. Main tasks, performed by BFF

- exchange of Authorization Code for tokens
- saving all tokens to protected cookies
- exchanges refresh tokens on new access tokens
- performing logout (closing the user session) and zeroing all tokens
- sending a request to get data to the Resource Server and adding access token to the Authorization header


THE CONCEPT OF A BACKEND-FOR-FRONTEND
Run the *Authorization Code* flow with client authentication 1
AUTHORIZATION SERVER
2 Issue access token and refresh token
FRONTEND
Cookie-based session
BACKEND
3 Proxy API requests with access token retrieved from session
API
The OAuth 2.0 client application
The client can follow best practices for backend applications (client authentication, sender constrained tokens, …)
7