Login & Authentication Workflows

FusionAuth

Auth. Built for Devs, by Devs.

Introduction

The Landscape

The landscape of applications today is broad. Similarly, the methods of authentication used is similarly varied. This section covers some of the login and authentication workflows used by applications today. These examples use FusionAuth as the IdP (identity provider), but any IdP could be used. You can use these articles to help architect and design the authentication system for your application.

Keep in mind that this list is not exhaustive. Most IdPs, including FusionAuth, are capable of other forms of login. Also, there are many login workflows that we do not cover here due to security concerns with those methods.

Also, these articles do not discuss the pros and cons of the different types of applications and which might be best for your needs. They focus entirely on authentication for applications. There are many articles available that cover different application types and why you should choose one over the other (such as the decision to use a native mobile application versus a responsive web application).

Definitions

Native Login Form

This is a form built directly into the application rather then leveraging an external login form such as OAuth

Application Backend

This is the backend of the application, not the IdP (i.e. not FusionAuth)

Traditional Web Application Authentication

Traditional web application authenticaiton use the request and response nature of the HTTP protocol along with the URL address of the browser to provide functionality to users. Many new web applications are still implemented using this method and many existing web applications use this pattern.

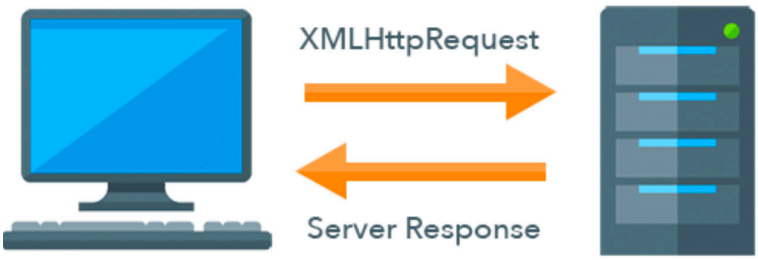
These applications load a web page by making a request from the browser to the server based on the URL address. The server responds to these requests with HTML, CSS & JavaScript. These documents are then rendered by the browser.

When the user clicks a link, button or submits a form, the browser makes a new request to the server and changes the URL in the address bar. The server handles the request and returns HTML, CSS & JavaScript that is rendered by the browser again. The browser renders the new documents.

There are only two HTTP methods supported by the browser in this style of application, GET and POST. The HTTP method informs the server what type of action the user took and how to handle the request.

Single-Page Application Authentication

Single-page web applications (SPA sometimes pronounced spa or S-P-A) Use the request and response nature of the HTTP protocol along with the URL address of the browser to provide functionality to users. Many new web applications are still implemented using this method and many existing web applications use this pattern.



Once the application is retrieved, it is started by the browser, normally by invoking a JavaScript framework that bootstraps and then begins to render the application. In many cases, the HTML of the application is produced by JavaScript and the browser updates the current document dynamically.

SPAs use JavaScript and call APIs to handle user interactions and input. For example, if a use clicks a link, the SPA might modify the document to render a different page or form. When a user submits a form, the SPA might call an API on the server.

APIs are invoked using the XMLHttpRequest functionality of browser JavaScript. This generally makes calls to REST APIs on the server. When the server responds, the JavaScript running in the browser handles the response and updates the document.

Since the browser is using XMLHttpRequest, it can invoke all of the standard HTTP methods including GET, POST, PUT and DELETE. The HTTP method helps inform the server what type of action the user took and how to handle the request.

It is important to note that while we recommend OAuth for SPAs, it causes the browser to close the SPA and navigate to the OAuth provider. Once the OAuth workflow is complete, the browser retrieves and starts the SPA again. This process might seem somewhat heavy, but the SPA should be cached in the browser making the second startup process much faster.

Native Mobile Application Authentication

Native mobile applications are usually installed via a store and installed on the mobile device (phone, tablet, etc). These applications are started by clicking the icon on the device. The device operating system then starts the application. Once the application is started, it renders its user interface.

Native applications often call APIs to handle user interactions and input. For example, if a use clicks a button or submits a form, the application might call an API on the server. This API might be called via HTTP or some other type of protocol. Often, native applications use various libraries for making API calls simpler.

Implementation Notes

Some experts recommend that native applications (including mobile apps) use OAuth's authorization code grant. This method works fine with many IdPs, including FusionAuth, but is not listed in this section because it is covered in the SPA and Webapp sections above. The only difference is that at the end of the OAuth workflow, the native application pulls the JWT and refresh tokens from the web-view.

Here are the options to providing login to traditional webapps.

Native login form to the application backend using JWTs and refresh tokens stored in cookies
Native login form to the application backend using sessions
Native login form to the application backend using sessions plus refresh tokens stored in cookies
(RECOMMENDED) OAuth 2 authorization code grant using JWTs and refresh tokens stored in cookies
(RECOMMENDED) OAuth 2 authorization code grant using sessions
OAuth 2 authorization code grant using sessions plus refresh tokens stored in cookies
OAuth 2 resource owner password credentials grant using JWTs and refresh tokens stored in cookies
OAuth 2resource owner password credentials grant using sessions
OAuth 2 resource owner password credentials grant using sessions plus refresh tokens stored in cookies

Here are the options to providing login to SPAs.

Native login form to the application backend using JWTs and refresh tokens stored in cookies
Native login form to the application backend using sessions
Native login form to the application backend using sessions plus refresh tokens stored in cookies
Native login form to FusionAuth using JWTs stored in local storage and refresh tokens stored in cookies
Native login form to FusionAuth using JWTs and refresh tokens stored in cookies
Native login form to FusionAuth (same domain) using JWTs and refresh tokens stored in cookies
(RECOMMENDED) OAuth 2 authorization code grant using JWTs and refresh tokens stored in cookies
(RECOMMENDED) OAuth 2 authorization code grant using sessions
OAuth 2 authorization code grant using sessions plus refresh tokens stored in cookies
OAuth 2 implicit grant using JWTs stored in cookies
OAuth 2 implicit grant using JWTs stored in local storage
OAuth 2 implicit grant using sessions
OAuth 2 resource owner password credentials grant using JWTs and refresh tokens stored in cookies
OAuth 2 resource owner password credentials grant using sessions
OAuth 2 resource owner password credentials grant using sessions plus refresh tokens stored in cookies

Here are the options to providing login to native mobile applications.

Native login form to the application backend using JWTs and refresh tokens
RECOMMENDED) Native login form to FusionAuth using JWTs and refresh tokens
OAuth 2 resource owner password credentials grant using JWTs and refresh tokens

FusionAuth

Auth. Built for Devs, by Devs.

FusionAuth is the authentication and authorization platform built for developers, by developers. For technical leaders creating products for external users, it solves the problem of building essential user security without distracting from the primary application.

Learn more at **FusionAuth.io**