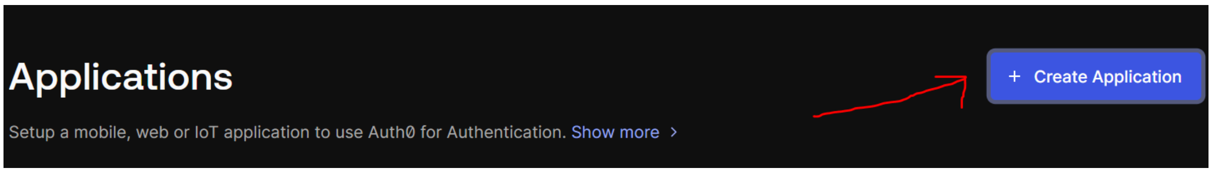
Auth0 Integration Guide

This guide will get you through the basics of how to set up auth0 on a single-page web server. After following this guide, you will have an example where you can log in and out of a webpage. This guide will be written in javascript.

1. Sign up for Auth0 and create app

For this guide to work you need to sign in at Auth0's website and create single-page web app. This link leads to their login page. You can log in however you fancy. Once in, head over to the navigation bar, and select 'Applications' then create a new application, which will be a Single Page Application. Its name is irrelevant.


Applications
Setup a mobile, web or IoT application to use Auth0 for Authentication. Show more ›
+ Create Application

2. Configure app

Once your app is created, you can go into its **Application Settings**. here you will find two important pieces of information:

- Your **Domain**
- Your **Client ID**

These will become important later. For now you need you need to configure your **Callback URLs, Logout URLs,** and **Allowed Web Origins.** You need to write 'http://localhost:3000' in all of these. Replace the port with the port your server is running on, but in my case it is port 3000.

3. Integrate auth0 into your app

Assuming you have a project running in your preferred choice of IDE, you need to install the necessary dependencies. Open up a terminal and write:

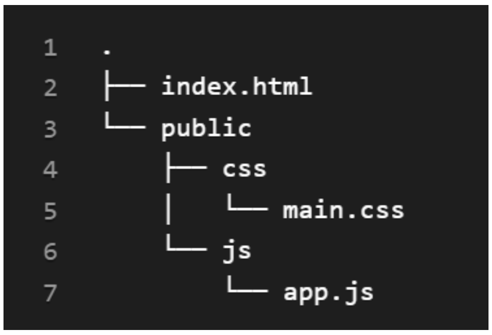
```
npm install --save @aut0/auth0-spa-js
```

This will install the auth0 SDK. Once this is done, you can import it either via a module-import in your application, or via a script in the HTML. For this example, we will be writing the html script, like this:

```
<script
src="https://cdn.auth0.com/js/auth0-spa-js/2.0/auth0-spa-js.produc
tion.js"></script>
```

4. Setting up the basic application.

This guide assumes your project runs on a structure similar to this:



```
.
├── index.html
└── public
    ├── css
    │   └── main.css
    └── js
        └── app.js
```

With a basic html page (including the script from above), a main.css and an app.js. We'll presume you're following from a brand new, empty project. Adjust it as you'd like for an existing project.

Let's start by fully visualizing this HTML-page we've been talking about:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>SPA SDK Sample</title>
    <link rel="stylesheet" type="text/css" href="/css/main.css" />
  </head>

  <body>
    <h2>SPA Authentication Sample</h2>
    <p>Welcome to our page!</p>
    <button id="btn-login" disabled="true" onclick="login()">Log
in</button>
    <button id="btn-logout" disabled="true" onclick="logout()">Log
out</button>

    <div class="hidden" id="gated-content">
      <p>
        You're seeing this content because you're currently
        <strong>logged in</strong>.
      </p>
      <label>
        Access token:
        <pre id="ipt-access-token"></pre>
      </label>
      <label>
        User profile:
        <pre id="ipt-user-profile"></pre>
      </label>
    </div>

    <script
src="https://cdn.auth0.com/js/auth0-spa-js/2.0/auth0-spa-js.production.j
s"></script>
  <script src="js/app.js"></script>
  </body>
</html>
```

You have the basic structure, two buttons for log-in and log-out, and a ‘hidden’ class which we will write in our .css soon. Once the application is finalized, you’ll be able to see the displayed credentials when you’re logged in.

Next we will write the .css file:

```
.hidden {
    display: none;
  }

  label {
    margin-bottom: 10px;
    display: block;
  }
```

This includes the ‘.hidden’ class mentioned before as well as rules for the label.

Next we’ll need to configure credentials. From the root folder, create a new file called ‘auth_config.json’ and write the following:

```
{
    "domain": "",
    "clientId": ""
  }
```

Replace the empty quotations with your domain and client ID found in your Application Settings (make sure to keep the quotations surrounding the strings).

Now we need to create the server itself. First things first, initialize your application by writing the following into your terminal:

```
npm init -y
```

This will initialize a new NPM project and allow us to install the necessary dependencies.

This server will use express, so install it.

```
npm install express
```

Once this is done we can build the server itself:

```
const express = require("express");
const { join } = require("path");
const app = express();

// Serve static assets from the /public folder
app.use(express.static(join(__dirname, "public")));

// Endpoint to serve the configuration file
app.get("/auth_config.json", (req, res) => {
  res.sendFile(join(__dirname, "auth_config.json"));
});

// Serve the index page for all other requests
app.get("/*", (_, res) => {
  res.sendFile(join(__dirname, "index.html"));
});

// Listen on port 3000
app.listen(3000, () => console.log("Application running on port 3000"));
```

This provides two endpoints:

- One which serves the information from our auth_config file.
- Another which serves the index page for all other requests.

Now to write the main functionality in our app.js.

```
// Empty variable to hold the auth0 object later.
let auth0Client = null;

// Function that fetches the information from auth_config
const fetchAuthConfig = () => fetch("/auth_config.json");

// New function that uses fetchAuthConfig to download its content and also
initializes the auth0 variable.
const configureClient = async () => {
    const response = await fetchAuthConfig();
    const config = await response.json();

    auth0Client = await auth0.createAuth0Client({
      domain: config.domain,
      clientId: config.clientId
    });
  };

// Call to initialize above function on page load.
```

```
window.onload = async () => {
    await configureClient();

  // update the UI state
  updateUI();

  const isAuthenticated = await auth0Client.isAuthenticated();

  if (isAuthenticated) {
    // show the gated content
    return;
  }

  // Check for the code and state parameters
  const query = window.location.search;
  if (query.includes("code=") && query.includes("state=")) {

    // Process the login state
    await auth0Client.handleRedirectCallback();

    updateUI();

    // Use replaceState to redirect the user away and remove the query
string parameters
    window.history.replaceState({}, document.title, "/");
    }
};

const updateUI = async () => {
  const isAuthenticated = await auth0Client.isAuthenticated();

  document.getElementById("btn-logout").disabled = !isAuthenticated;
  document.getElementById("btn-login").disabled = isAuthenticated;

  // Add logic to show/hide gated content after authentication
  if (isAuthenticated) {
    document.getElementById("gated-content").classList.remove("hidden");

    document.getElementById(
      "ipt-access-token"
    ).innerHTML = await auth0Client.getTokenSilently();

    document.getElementById("ipt-user-profile").textContent =
JSON.stringify(
      await auth0Client.getUser()
    );

  } else {
    document.getElementById("gated-content").classList.add("hidden");
  }
};

// Function for login, called by the login button defined in html.
const login = async () => {
    await auth0Client.loginWithRedirect({
      authorizationParams: {
        redirect_uri: window.location.origin
      }
    });
};

// Function for logout, called by logout button once user is authenticated.
const logout = () => {
    auth0Client.logout({
      logoutParams: {
        returnTo: window.location.origin
      }
    });
};
```

Once this is done, you should be able to start your server through the terminal command:

```
node server.js
```

Access http://localhost:3000 and view the finished work. You'll now be able to do the following:

- Press the 'login' button to either login as an existing user or create a new account.
- View your user profile and access token once you're logged in.
- Log out via the 'Log out' button.

For further information and details, consult auth0's official documentation for a single page application in javascript.