web10 wapi.js SDK

back to web10 auth page

Installation

wapi.js is the javascript file containing the web10 developers SDK.
the file can be found at : https://auth.web10.app/sdk/wapi.js

```
<!-- Installing using CDN -->
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script src="https://auth.web10.app/sdk/wapi.js"></script>
```

wapi.js relies on axios to make web10 requests, which is included in the above CDN links.

Initialization

in order to use the web10 SDK, the main SDK object needs to be initialized by the developer.

function	description
wapiInit(authUrl)	returns a wapi object registered to handle web10 authentication at the auth portal of the given authUrl.

```
//initialize a wapi object registered for auth with auth.web10.app
const wapi = wapiInit("https://auth.web10.app")
```

Authentication

once the wapi object is initialized, it provides a variety of functionalities for managing authentication and credentials.

function	description
wapi.isSignedIn()	returns a boolean : whether the app is signed in.
wapi.signOut()	signs the app out.
wapi.openAuthPortal()	opens the registered web10 auth portal.
wapi.authListen(setAuth)	listens for the auth portal to send a login token, and triggers the inputted callback function [setAuth]
wapi.readToken()	reads the data fields of the web10token stored in the wapi object. if wapi isn't logged in and the token is null, wapi.readToken() returns null.

function	description
wapi.getTieredToken(site,target)	mints a token for a given site and web10server using the token stored in wapi. returns an axios promise with response data being the token[as JWT string] on success.

Authentication - Hello World Demo

Below is an example of some html and javascript utilizing all of the above authentication functionality to handle login for a simple hello world app. **Demo Link**

```
<html>
    <!-- index.html -->
    <body>
        <button id="authButton">
            log in
        </button>
        <p id="message">
            app not started
        </p>
    </body>
    <!-- Installing using CDN -->
    <script src="https://unpkg.com/axios/dist/axios.min.js" ></script>
    <script src="https://auth.web10.app/sdk/wapi.js" ></script>
    <script src="script.js"></script>
</html>
```

```
/* script.js */

//initialize a wapi object registered for auth with auth.web10.app
const wapi = wapiInit("https://auth.web10.app")

// make the auth portal open when the log in button is pressed
authButton.onclick = wapi.openAuthPortal

// callback function that initializes the app on web10 login
function initApp(){
    // make the logout button handle logouts properly on login
    authButton.innerHTML = "log out";
    authButton.onclick = () => {
        wapi.signOut();
        window.location.reload();
    }
    // simple hello world app, saying hello to the user
    const t = wapi.readToken()
    message.innerHTML = `hello ${t["provider"]}/${t["username"]},<br>`
}

// either initialize the app if logged in, wait for authentication to do so.
if (wapi.isSignedIn()) initApp()
else wapi.authListen(initApp)
```

web10 Services

a web10 service is a managed MongoDB collection provided by a web10 provider

web10.app services are hosted at :

On Your Own Terms

> users start new web10 services by accepting SIRs [service initialization requests]
>
> users accept or deny changes to terms of service through SCRs [service change requests]
>
> > users can change their terms of service in the web10 authentication portal at any time.

User Owned Service Management

function	description
wapi.SMROnReady(sirs,scrs)	adds an event listener that waits for the authentication service to send a ready signal. when the authentication service is ready, wapi sends a service modification request [SMR]. an SMR consists of list of service initialization requests [SIRs] and a list of service change requests [SCRs]
wapi.create(service,query,username,provider)	Runs a MongoDB create on the web10 service at provider/{username}/{service}, and returns the result as an axios promise.
wapi.read(service,query,username,provider)	Runs a MongoDB read on the web10 service at provider/{username}/{service}, and returns the result as an axios promise.
wapi.update(service,query,update,username,provider)	Runs a MongoDB update on the web10 service at provider/{username}/{service}, and returns the result as an axios promise.
wapi.delete(service,query,username,provider)	Runs a MongoDB delete on the web10 service at provider/{username}/{service}, and returns the result as an axios promise.

Demo - Note App

Below is an example of some html and javascript utilizing all of the above user owned service management functionality to make a basic notes app. **Demo Link**

```
<html>
<!-- index.html -->
<body>
    <button id="authButton">
        log in
    </button>
    <p id="message">
        app not started
    </p>
    <div>
        <textarea id="curr" placeholder="write a note here"></textarea>
        <button onclick="createNote(curr.value)">create note</button>
    </div>
    <div id="noteview"></div>
</body>
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script src="https://auth.web10.app/sdk/wapi.js"></script>
<script src="script.js"></script>
</html>
```

```
/* script.js */

//conventient failure messages
const Fs = ([cF, rF, uF, dF] = ["create", "read", "update", "delete"].map(
    (op) => `failed to ${op} note[s]`
  ));

/* wapi setup */
const wapi = wapiInit("https://auth.web10.app")
const sirs = [
  {
    service: "web10-docs-note-demo",
    cross_origins: ["auth.web10.app","jacobhoffman.tk"],
  },
];
wapi.SMROnReady(sirs, []);
authButton.onclick = wapi.openAuthPortal;

function initApp() {
  authButton.innerHTML = "log out";
  authButton.onclick = () => {
    wapi.signOut();
    window.location.reload();
  };
  const t = wapi.readToken();
  message.innerHTML = `hello ${t["provider"]}/${t["username"]},<br>`;
  readNotes();
}

if (wapi.isSignedIn()) initApp();
else wapi.authListen(initApp);
```

```
/* CRUD Calls */
function readNotes() {
  wapi
    .read("web10-docs-note-demo", {})
    .then((response) => displayNotes(response.data))
    .catch(() => (message.innerHTML = rF));
}
function createNote(note) {
  wapi
    .create("web10-docs-note-demo", { note: note ,date:String(new Date())})
    .then(() => {
      readNotes();
      curr.value = "";
    })
    .catch(() => (message.innerHTML = cF));
}
function updateNote(id) {
  const entry = String(document.getElementById(id).value);
  wapi
    .update("web10-docs-note-demo", { _id: id }, { $set:{note: entry }})
    .then(readNotes)
    .catch(() => (message.innerHTML = uF));
}
function deleteNote(id) {
  wapi
    .delete("web10-docs-note-demo", { "_id": id })
    .then(readNotes)
    .catch(() => (message.innerHTML = dF));
}

/* display */
function displayNotes(data) {
  function contain(note) {
    return `<div>
                <p style="font-family:monospace;">${note.date}</p>
                <textarea id="${note._id}">${note.note}</textarea>
                <button onclick="updateNote('${note._id}')">Update</button>
                <button onclick="deleteNote('${note._id}')">Delete</button>
            </div>`;
  }
  noteview.innerHTML = data.map(contain).reverse().join(`<br>`);
}
```

thanks for making it to the end :)