Codecolab

Description

This project is a plugin that has been built inside OS.js. MULE itself is a plugin built on top of OS.js's framework with multiple packages that allow it to create a learning environment for Maynooth university students. Codecolab (the project itself) has been installed inside MULE as a plugin to allow for anyone using the MULE environment to have the ability to collaboratively edit code documents in real time with their friends. Once MULE has been installed on your machine using Docker, MULE should be running on port 80. Codecolab will be located in the menu on the top left under the 'Other' menu item.

Impotant Note

The repository provided will not show you a full commint history of the project from start to finish this is because codecolab was developed in the OS.js repo and then when it was fully developed and tested it was copied and pasted into the MULE repository. Below is a link to the repo with the full commit history

```
https://github.com/aking-a/OS-App-FYP
```

Installation

This will guide you to enable you to be able to install and build MULE with the codecolab plugin already installed. Please note that wsl2 and docker is slow and it will take time for the application to work properly if you are using a windows system. This is a known issue and is caused by windows and wsl2. I was unable to find a solution to this so I used ubuntu to run and build the MULE container, I would recommend this approach to run and build the application if possible.

1. Clone the repository:

```
git clone https://github.com/aking-a/mule-code-system
```

2. Install the docker desktop:

```
https://www.docker.com/products/docker-desktop/
```

3. Enable Linux subsystem wsl2 if you are using windows:

```
https://learn.microsoft.com/en-us/windows/wsl/install
```

4. Build and Run the container:

```
docker-compose up -d
```

5. Open your browser on localhost:

```
http://localhost:80/
```

Usage

1. *Login:* If the container has been built correctly you will be greeted with a login page. In the MULE repo navigate to the 'index.html' file in the .login folder.

2. *Open with Live Server:* Install the Live Server extension for visual studio code and open the index.html file. You should see this page below. Do not change any of the variables! Click the **build login request button** and then click the **Login with LTI button**.


3. *Logged in:* You should be navigated to the MULE web desktop which should look like the screenshot below. If you have not been relocated to the MULE try opening localhost in a new tab.

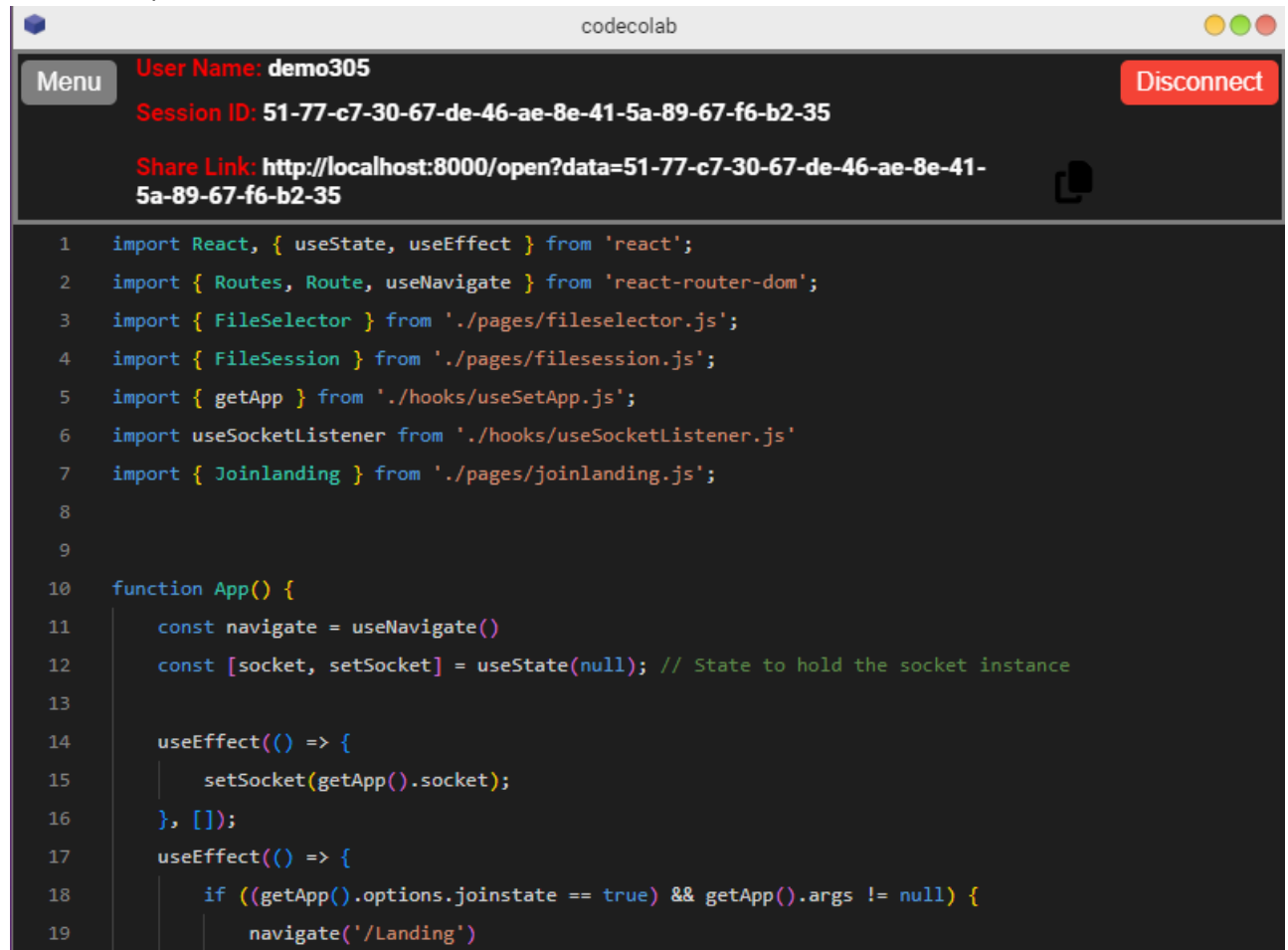4. *Open Codecolab:* Navigate to the top left menu button click on it then click the **Other** menu item then **codecolab**. The application landing page should open and look like the screenshot below.
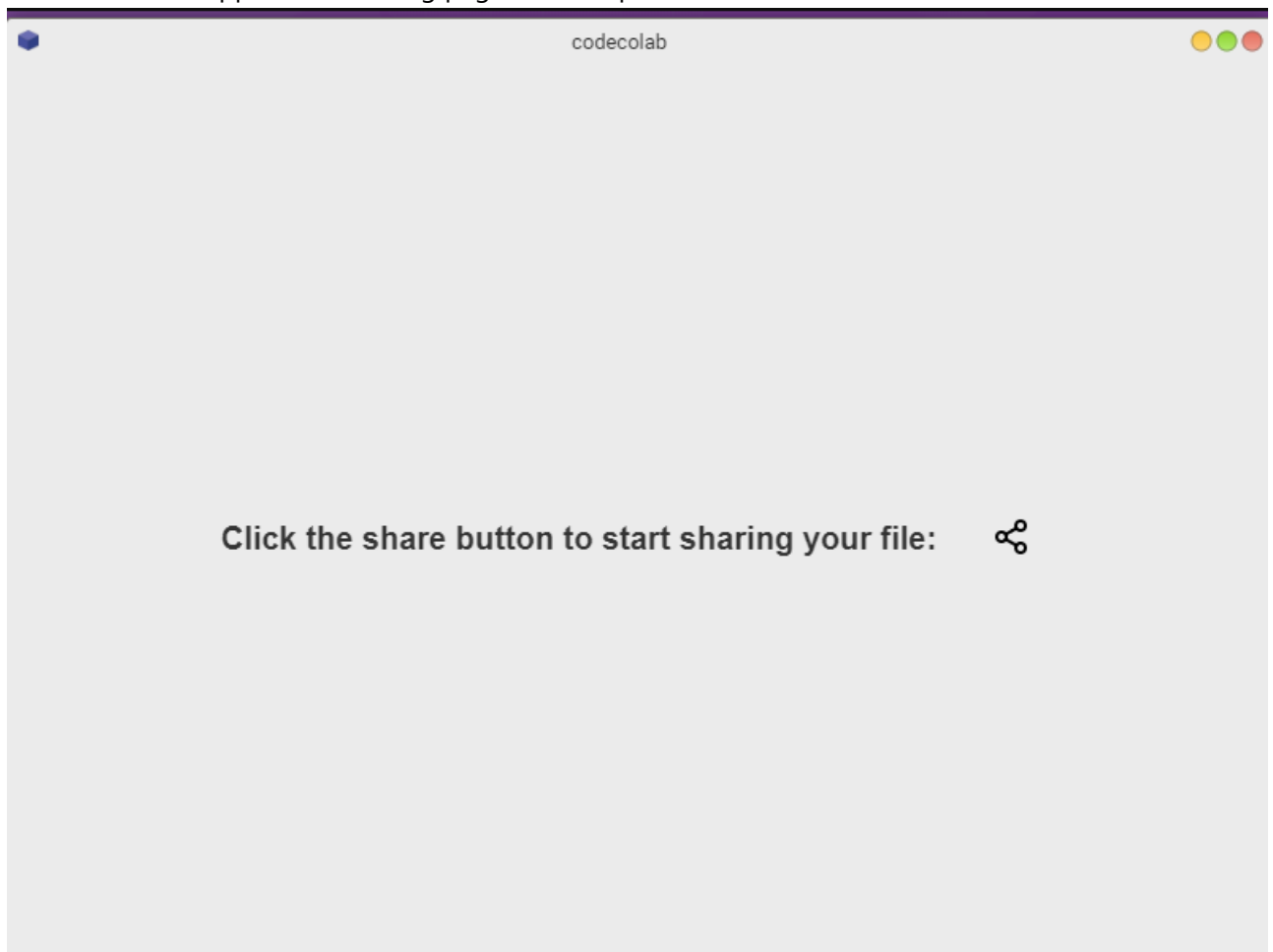
5. *Share file:* Click the share file button as instructed and select a file from MULES virtual file system. You should then be navigated to the main window which is the code editor and it should look like the

screenshot provided.
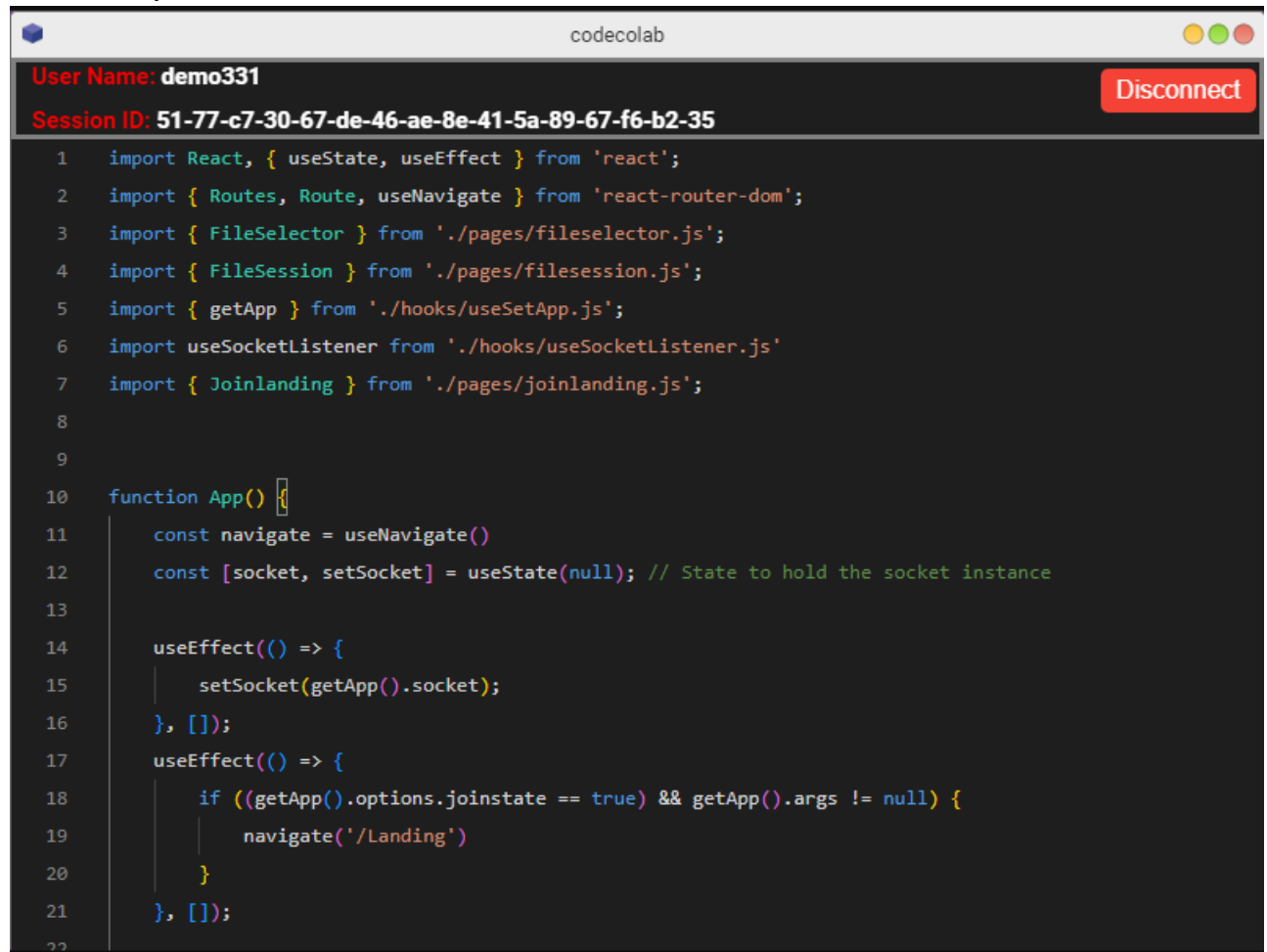
6. *Other Users:* Using the share link you can copy and paste this into a new tab and it will join that session that has been created. You can now start editing the document on either tab and you should see the changes be reflected in both tabs. Below is a screenshot of what the user that joined the session through the link should look like. Please note that the menu for other users that join the session is

disabled so you will not see it on the second tab.

```
codecolab                                                    ○○●

User Name: demo331                                              Disconnect
Session ID: 51-77-c7-30-67-de-46-ae-8e-41-5a-89-67-f6-b2-35

1    import React, { useState, useEffect } from 'react';
2    import { Routes, Route, useNavigate } from 'react-router-dom';
3    import { FileSelector } from './pages/fileselector.js';
4    import { FileSession } from './pages/filesession.js';
5    import { getApp } from './hooks/useSetApp.js';
6    import useSocketListener from './hooks/useSocketListener.js'
7    import { Joinlanding } from './pages/joinlanding.js';
8
9
10   function App() {
11       const navigate = useNavigate()
12       const [socket, setSocket] = useState(null); // State to hold the socket instance
13
14       useEffect(() => {
15           setSocket(getApp().socket);
16       }, []);
17       useEffect(() => {
18           if ((getApp().options.joinstate == true) && getApp().args != null) {
19               navigate('/Landing')
20           }
21       }, []);
22
```

Contributing

Please follow these steps to contribute to the project: I personally used the OS.js documentation to create a application inside OS.js and then I copied and pasted that application into MULE's directory specifically in **mule/src/packages**. I found it easier to build and test the application without Docker and then build the container once the finished product had been realised.

1. *Building an application in OS.js:* Go to OS.js and clone its repository the instructions to set it up on your own machine are explained there. Use the non-docker setup instructions!

   ```
   https://manual.os-js.org/
   ```

2. *Copy codecolab and build:* Copy the codecolab folder from the MULE repo provided above and copy it into **src/packages** in the OS.js repository on your machine. Assuming you have OS.js running on your local machine stop it and run the commands below. Do not copy the dist and node_modules package from codecolab!

   ```
   cd src/packages/codecolab
   ```

```
npm install
```

cd back out of src/packages/codecolab

```
npm run package:discover
```

```
npm run build
```

3. *Serve and Watch:* Once the build command has run successfully codecolab will now be installed as a package. Use the **npm run serve** command to start OS.js it should be running on **http://localhost:8000/**. *(An important note to mention is the invite link generator does not take into account what port or URL the server is running on so please change this in the source code. The link generator is located in codecolab /server-modules/newsession.js line 15.)*. To run a watch on codecolab package use the commands provided below.
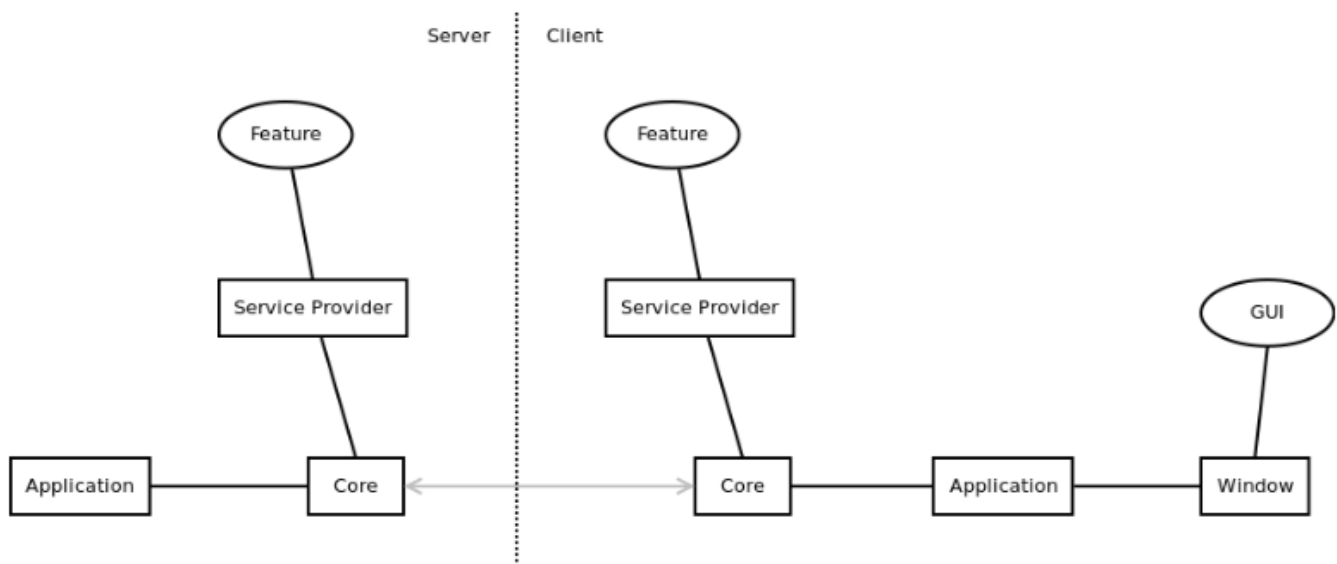
```
cd src/packages/codecolab
```

```
npm run watch
```

Design Explained

This section will cover OS.js's overall architecture and codecolabs files and structure along with libraries and packages used.

This is a screenshot from OS.js that provides a high-level overview of how it's framework works

This image shows how applications are built inside a window and contained within OS.js web desktop.

Codecolab itself is an application and through the use of react I was able to render a react app inside one of these windows. The react app navigates to its different routes using a react library called BrowserRouter. The OS.js itself along with all of its packages is bundled and served using webpack therefore *I would recommend researching react and webpack before attempting to make changes to this repo and of course research the OS.js documentation.*

codecolab file structure and file functionality:
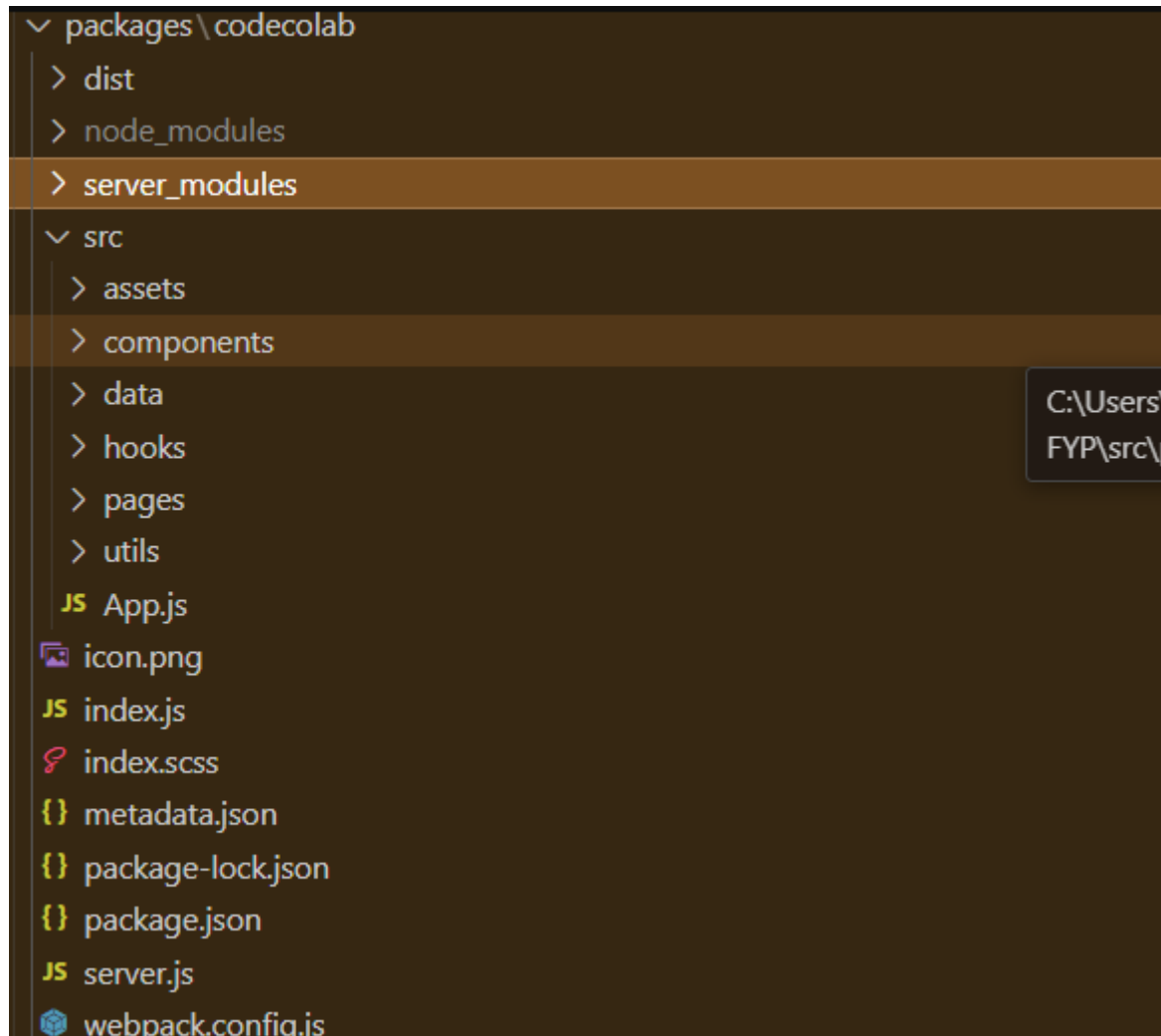
Screenshot of codecolab's file structure

Top Level As seen in the screenshot above the top level contains the folders assets, components, data, hooks, pages, utils.

- *Assets* contains all the styling for the pages and components in the app.
- *components* contains all the components for the app such as the dropdown menu and buttons.
- *data* contains all the session data such as the file being used and the Monaco editor options.
- *hooks* contains all the react hooks used in the app.
- *Pages* contains the pages/routes that the application uses
- *Utils* contains important functions that are used in different parts of the application like on click events

packages used: These are the main packages used to create codecolab.

- *Webpack* This package is a key part of OS.js and is responsible for bundling and serving MULE/OS.js to the client.
- *React-Monaco-editor* This is a IDE that can be used on a website. It is the same IDE that VSCODE uses and comes with a lot of helpful built in functionality such as *onDidModelChangeContent()*.
- *Babel-loader* This is responsible for transpiling the js files.
- *CSS AND SASS loader* This is responsible for transpiling the css files in the project.
- *React* This is a well known UI package that is used for creating SPA's.
- *ws* This a basic websocket package that allows for web socket servers to be created in addition to creating a connection between client and server.

File Description

Each file will now be explained:

- **index.js:** This is the entry point in the application and it is where the window is created. The react app is imported to this application and is rendered inside the window that has been created. The react app is attached to the *$content* element. The apps relevant variables are stored inside AppData class object for later use. Below is a screenshot of the code performing this action. The socket connection to OS.js is also setup here.

```
// Create  a new Window instance
//This is all included when you run OS.js npm run make:package
var win = proc.createWindow({
  id: 'codecolabWindow',
  title: metadata.title.en_EN,
  icon: proc.resource(proc.metadata.icon),
  dimension: { width: 800, height: 600 },
  position: { left: 700, top: 200 }
}).on('destroy', () => proc.destroy())

//Afer a window has been created we use the already connected socket that is serving the OS.js features top us with a modification to the ws url which points
//to our appliaction server

const socket = proc.socket('/socket')

//a json that stores all the relavant app instances for use in the app
const app_data = new AppData(win, args, options, proc, osjs, socket, core)
useSetApp(app_data)

//getting the $content of the window to to render attach my react app to it so os.js window can render it
const $content = win.$content;
const root = createRoot($content);
//here is where we render the react app to the window, BrowserRouter is used to enable routing in the app
win.render(root.render(<BrowserRouter><App /></BrowserRouter>));
```
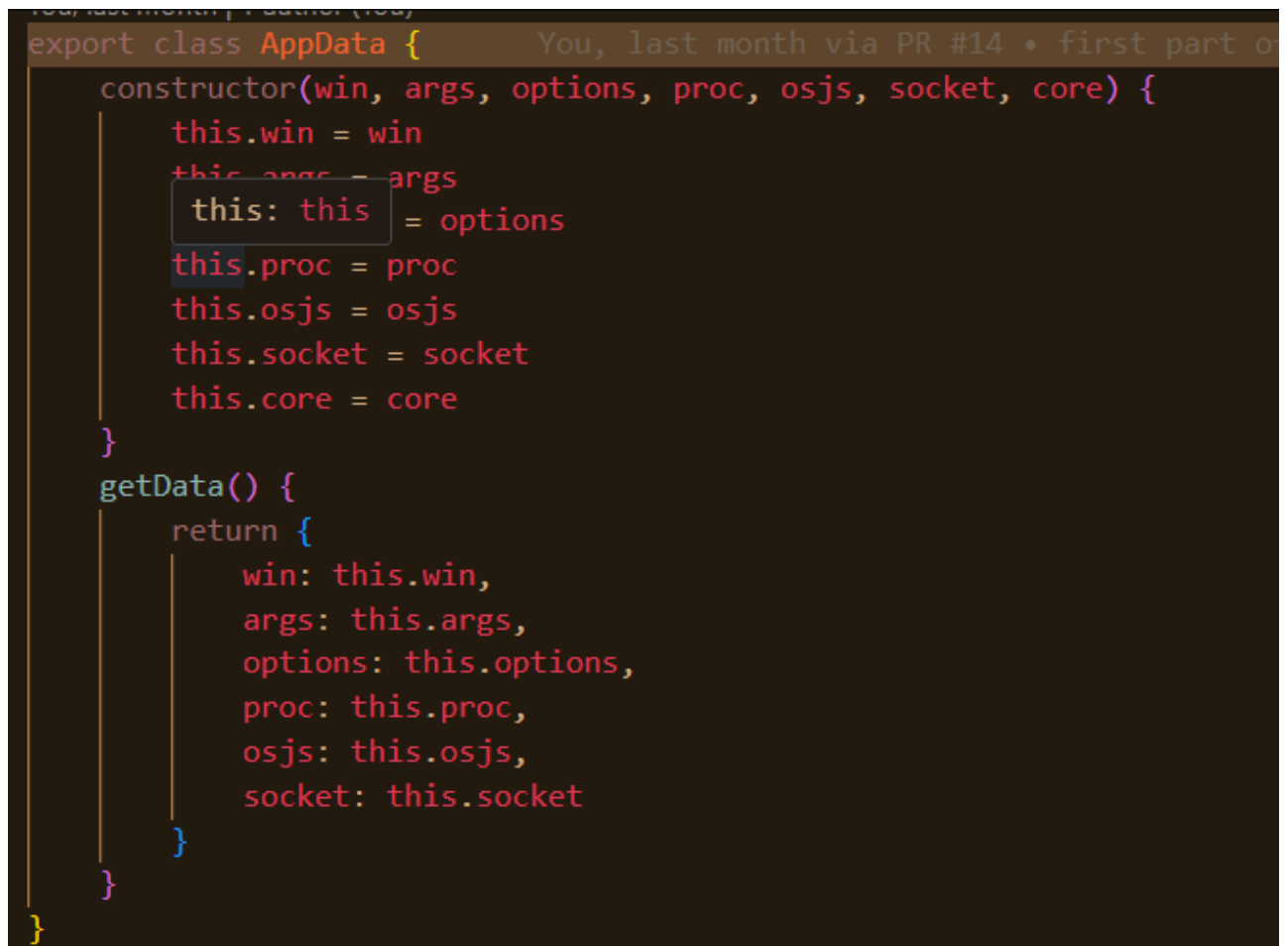
- **App.js:** This is the main entry point for the react app. The '/' route is served unless it is a client connecting through on invite link then 'Landing' will be served to the client. The is all done through the Browser Router package and the navigate function.

- **Disconnect.js:** This is the disconnect button component. Its logic is handled in the *handleDiconnect* function which basically sends a ws Json to the server telling it to remove that websoket from the session. It also uses the chakra UI library to help position it.

- **dropdown.js:** This is a component that constructs the dropdown menu for the app. The dropdown contains two options, save and user list. Save simply saves the file and user list lists the user in the session. The selection of a item in the menu is handled in handleSelect function. This component also sets up the usernames UseState array which keeps track of the users usernames that are connected to the session.

- **left_join_alert:** This is the popup component that appears when a user joins or leaves a session.

- **userlistwindow:** This is the component that is rendered inside the user list window. It is responsible for displaying the current list of users in a session.

- **appdata.js:** This is a class that is created in *index.js* and stores all the app data such as core for later use in other files. Below is a screenshot of its contents.

- **editoroptions.js:** This is the options used for Monaco editor.

- **file.js:** This is a class object that stores the file path data and the content of the file after it has been loaded from OS.js's *vfs*.

- **sessionclass.js:** A lot of variables are stored in the session class as you might have noticed. This class object is created before the user is navigated to the main page and is interacted with throughout the app. It is basically makes the variables within it globally accessible to any file in the application. Below is a screenshot of the class.

```
export class Session {
    constructor(file,socket,username) {
        this.file = file
        this.socket = socket
        this.username = username
        this.language = null
        this.sharelink = null
        this.editorRef = null
        this.sessionID = null
        this.code = null
        this.showPopup = null
        this.popupMessage = null
        this.isVisible = null
        this.userlist= null
        this.usernameslist = null
        this.itemlist = null
        this.ProgrammaticChange = false
        this.monaco = null
        this.lockedlines = new Set()
        this.curline = null
    }
}
```

- **useActionListener.js:** This file uses a hook that uses useEffect() from react to listen for changes in the document that Monaco has loaded. Once a change has been detected it packages that change and sends it to the server via the clientChange() function.

- **useDidMountListener.js:** Listens for the Monaco editor DIDMount and sets up the session by setting the userID code and other variables.

- **useListListener.js:** Listens for changes being made to the usernames list and applies the updated list to the itemlist in the *userlistwindow*. This causes the component to re-render the list when a new user joins or leaves.

- **useSetApp.js:** Stores the AppData class object so it can be retrieved from any file.

- **useShowPopupListener.js:** Shows the Popup for 3 seconds then sets the UseState to false to hide the popup.

- **useSocketListener.js:** Listens for incoming socket messages from the server and handles them.

- **fileselector.js:** This file is the landing page for creating a session. It just loads the svg icon and text. The click event is handled by the *clickEvent()* function.

- **filesession.js:** This is the main page in the application and all the components are useHooks are imported and used here. The react-monaco-editor is used here and is rendered in this section. A lot of the useStates are also set up and used here.

- **joinlanding.js:** This is the alternate route that users that are joining a session take. The reason behind this was to skip the file selector as users joining a session are not allowed to select a file. Additionally it displays a flashing loading text while the user is joining the session.

- **clickevent.js:** This is responsible for handling the clickEvent from *fileselector*. It loads and stores the selected file data in the session class object.

- **renderlist.js:** Creates a new OS.js window and renders the list component inside it.

- **handlechnages.js:** Applies the incoming changes from the server to the document.

- **handledidmount.js:** Handles the did mount of the Monaco editor.

- **handleReadOnlyLines.js:** Makes lines read only by moving the cursor away from lines that are in the set of locked lines.

- **handlesocketoutgoing.js:** Handles the outgoing events that are sent to the websocket server.

- **getusername.js:** Gets the username that has been used to log into MULE or OS.js and applies a random number to it and returns it.

- **updateList.js:** Adds and removes usernames from the user list *(the list of users in the session)*.

- **getSession.js:** Stores and Returns the *sessionclass* object. It also has a function called *Terminate()* that destroys the object.

- **handleSelect.js:** Handles the selection of a menu item from the dropdown menu.

- **openfile.js:** This file is responsible for loading a file from OS.js's *vfs* using promises.

- **savefile.js:** This file saves a file to OS.js's file system using the file data from the *fileclass* object.

- **server.js:** This server handles all the connected websockets and broadcasts changes to connected clients. Each session is stored in a *createnewsession* class. This class stores all of the relevant data for a session such as the file being edited and a list of the websockets connected to that session. Then each of these classes is stored in a Json that is identifies using a crypto generated key which serves as it's unique identifier. All of the other cases are handled here aswell such as disconnect broadcast and acquiring locks etc.

- **onDisconnect.js:** This handles the broadcast when a user disconnects or loses connection to the session. It also deletes the session and forces connected users to leave if the user that left is the admin. It will also remove connected clients from the client array if they are not an admin.

- **serverDocEditor.js:** This is responsible for modifying the document state on the server.

- **tryoperation.js:** This file broadcasts the lock and release of a line.

- **linelock.js:** This is the logic that is responsible for locking the line.

- **newsession.js:** This creates an object that stores the session data. A new session object is created each time a user creates a new file sharing session. It also has *createShareLink()* and *getLanguage()* which is responsible for creating the share link and extracting the language of the file from its extension.

- **LinkHandler.js:** This file is contained outside the codecolab package and is used to handle the /open route for the invite link. It basically serves the extracts the sessionID passed through the link data and serves the index html of mule. It will then execute the javascript to open codecolab on the sessionID that was ectracted from the link. This service provider is always active on the server side. It was created through *npm make:serviceprovider*. This command is from the OS.js documentation and more details on this can be found there. Here is a screenshot of the LinkHandler provider.

```javascript
//This html was given to me by the OS.js team someone in thw OS.js community that helped me with thi
app.route('get', '/open', (req, res) => {
  let data = req.query.data;
  let ans = decodeURIComponent(data);
  let domain = 'http://'+req.get('host');
  let page = `<!DOCTYPE html>
      <html lang="en">
      <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Dynamic Page</title>
      </head>
      <body>
        <script>
          function loadPage(ans) {
            history.pushState(null, null, '${domain}');
            fetch('/index.html')
              .then(resp => resp.text())
              .then(hdoc => {
                const base = new URL('/', location);
                hdoc = hdoc.replace('<head>', '<head><base href="' + base + '">');
                document.open();
                document.write(hdoc);
                document.close();
                // update config conforming to document.baseURI
                window.webpackJsonp = [[
                    [/*anonymous chunk*/],
                    {
                        'rwcfgurimod': (m, e, r) => {
                          const cfg = r('./src/client/config.js');
                          const href = document.baseURI;
                          const pathname = new URL(href).pathname;
                          (cfg.http ||= {}).public = pathname;
                          (cfg.http ||= {}).uri = href;
                          (cfg.ws ||= {}).uri = href.replace(/^http/, 'ws');
                        }
                    },
                    [["rwcfgurimod", "osjs"]]
                ]];
                // run package
                const groups = [], cb = {};
                const { group, groupEnd } = console;
                const delay = (f) => f && setTimeout(f);
                console.group = (n) => (groups.push(n), group(n));
                console.groupEnd = (n) => (n = groups.pop(), groupEnd(), delay(cb[n]));
                cb['Session::load()'] = () => {
                  const name = 'codecolab';
                  const args = ans;
                  const options = {joinstate:true};
                  OSjs.run(name, args, options);
                };
              });
          }
          loadPage('${ans}');
        </script>
      </body>
      </html>`
  res.send(page);
```