Integration Training Environment

This application is designed to mimic an ISV's own application, though it is highly simplified for training purposes.

It may take you a while to figure out everything that is happening in this application. That is okay, the learning experience will be invaluable!!

Application Framework Overview

Most modern web applications are split into two components:

Front End (Client Side) – JavaScript that is executed on the end user's web browser. This is the code that renders the webpage a user sees when navigating. This code will send requests for information to the "back end" server, and render the results of those on the page.
In Yellowfin – Backbone
Training Application – React
Other Frameworks – Vue.js, Angular

Back End (Server Side) – The webserver. This is typically where the bulk of the complex processing happens. This receives requests for data from the front end client, and sends back the appropriate info. There are a wide variety of languages that can be used for this, but the most common are Java, .Net(C# or C++), and Python.
ln Yellowfin – Spring written in Java – the Yellowfin backend is what communicates with client's reporting database and yellowfin's own configuration database.
Training Application – this is a framework called Django written in Python

The primary purpose of the backend in this training application is to send REST API calls to Yellowfin. To retrieve things such as the access/refresh tokens.

Key learning resources for each of the frameworks used in this:
React - https://reactjs.org/tutorial/tutorial.html
Django - https://docs.djangoproject.com/en/4.0/intro/tutorial01/

Installation

Install Python (3.7+) – https://www.python.org/
Install NodeJS - https://nodejs.org/en/download/
Install Visual Studio Code or your favorite IDE - https://code.visualstudio.com/

Copy the entire application folder somewhere onto your computer. Or grab the git Repo:
https://github.com/hannsta/YFIntegration

Open VS Code and open the entire application folder (Open Folder)

Open a new terminal in VS Code

```
cd C:/…/integrationenvironment
```

Install the python libraries:

```
pip install -r requirements.txt
```

Change directories into the backend:
```
cd backend
```

Run the Django Server:
```
python manage.py runserver
```

If this worked you will see:
```
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Open a new command prompt and navigate to the frontend folder
```
cd frontend
```

Start the Node application:
```
npm start
```

If you are on a mac, and see a permission denied error you may need to use:

```
sudo chmod +x node_modules/.bin/react-scripts
```

If this worked you will see:
```
Starting the development server...
Compiled with warnings.
```

You will access the environment via localhost:3000.

Don't forget to set the allow origins parameter to include localhost:3000, or simply a blank space as the first entry (make sure to click add another so there are 2 in total) which will act as a wildcard,

The Django environment will be running on localhost:8000, though you will not need to visit this except possibly for debugging purposes.

Testing the environment

If everything worked correctly you should now see the inside of the application!

Test the different buttons. Go through the refresh->access->login flow to authenticate your user, and click the "reports" or "dashboards" button.

Notice that the code section is populated with the JS and HTML that Yellowfin needs to render a report. You can change this code while on the page, and it will automatically refresh in the right hand container.

You may not see a report render on the right side of your screen. If this is the case, you will likely need to make sure:

1. the UUID of the report you are trying to embed exists in your Yellowfin
2. You configured Yellowfin with the appropriate CORS settings
3. You have configured the application to point to the correct Yellowfin instance (note again that we have hard coded localhost:8080, so if this is not your Yellowfin you will need to change it)

Debugging:

As you modify the code you will need to learn where errors are originating. Anything wrong with your JavaScript code (React front end) will show up in the Browser's dev console.

Anything wrong with the server, will show up in the command prompt where you started the server.

Changing the Yellowfin Credentials / Host

If you installed Yellowfin on a different port, or have changed the admin credentials, you may see errors right away indicating that the port/host/admin credentials we are trying to access Yellowfin with are not working properly. All of this can be adjusted.
Backend: Open the /backend/yellowfinintegration/yellowifn.py file, note the variables at the top of the page.

```
YELLOWFIN_URL = "http://localhost:8080"
ADMIN_USER = "admin@yellowfin.com.au"
ADMIN_PASSWORD = "test"
```

Frontend: Open the /frontend/src/codepen.js file, note the variables at the top of the page.

```
const YELLOWFIN_URL="http://localhost:8080"
```

Challenge: Note that this in production you would not have these hard coded. Adjust the code to pull these values from environmental variables or a configuration file.

Adding a new Yellowfin REST API call

Open the `/backend/yellowfinintegration/yellowifn.py` file

Currently there are 4 distinct methods in here, each focused around a REST API call to the Yellowfin instance. The first 3 go through the authentication flow specifically for the admin user. Obtaining the Refresh, Access, and Login tokens. The fourth is a call to obtain a list of signals available to the user (`get_signals`).

Adding a new REST call is very easy. We can simply create a new method in this file by copying an existing method. Depending on which API you are interested in using, you may want to leverage different elements of different calls. Notice the first 3 calls are all POST requests, while the signal call is a GET request. Additionally, notice that the first call contains data in the request body, while the last 3 simply contain a header.

Add an endpoint to our backend for the API call

Add a new URL to the urls.py file

Now that you have made an API call, its time to render that in the webpage. To do this, we need to create a new endpoint on our backend for the frontend to call.
Open the /backend/backend urls.py file

This defines the various URLs that are accessible on the backend. Notice that for each of the API calls defined in the previous step we already have a URL defined here. To add a new URL simply copy one of the lines in the urlpatterns, and adjust each of the words to reflect your new api:

```
path('access_token/', views.get_access_token, name='access_token'),
```

Add a new "view" to the views.py file

Open the /backend/yellowfinintegration /views.py file

This defines how each endpoint is handled when a request is made. Note again that for each of our API calls we have a corresponding method in the views file. Notice again how we have a method here for each API call and corresponding URL. To add your new call simply copy one of these methods and adjust the names to point to your appropriate API call.

```
get_access_token():
```

We now have everything we need on the server side. After saving all of your files you should

Call the new endpoint from the front end

Open the /frontend/src/Codepen.js file

Find the set of calls corresponding with the API calls.

```
function getAccessToken(){
  fetch('access_token?refresh_token='+encodeURIComponent(refreshToken))
    .then(response => response.json())
    .then(data => {
      setAccessToken(data)
      toast("Access Token Obtained: "+data)
    });
}
```

There are a few things happening here.

Fetch is the method in react that is used to make API calls. You can see that we are making a call to the URL defined in the urls.py file. Additionally we are defining a parameter in the url called refresh_token as this is required by the access_token api call.

The refresh token and access token are both stored using a React concept called “State”. Essentially, this is a set of remembered variables that we can store data in, and retrieve data from. State is specific to “components” in react. So as you extend this app it will be important to gain a stronger understanding of how state works. For this purpose, everything is done out of one primary component so we don’t need to know too much.

At the top of the page you will see two lines that define both refresh and access token variables, as well as a method that can be used to set each of them.

```
const [accessToken, setAccessToken] = useState('')
const [refreshToken, setRefreshToken] = useState('')
```

To properly add our own api call we will need to set a state variable for the data retrieved from the API. For now, we can just use the setHTML state method to display it on the page. Refer to get_signals to see how this is done for the signal API call.

Instead of:
`setAccessToken(data)`
Use:
`setHtml(JSON.Stringify(data))`

Finally, its time to add a button that our users can click to make the call itself! Scroll down the bottom of the Codepen.js file until you find the HTML defining the header. Each button is defined using the syntax shown below:

```
<div className="navOption" onClick={getSignals}>Signals</div>
```

To add your own, simply copy one of these lines and change getSignals to point to the method you created above to fetch data from the backend.

Next Step Challenges:

There is so much more we can do here. If you are looking to extend your knowledge, I would recommend trying the following two exercises:

Create a list of dashboards for the user. On click, switch which dashboard is rendered.

Add user authentication. Right now we are using a hardcoded admin, but in practice there will be a large number of users who each need to be authenticated individually.

Add client organizations - you will need to add a backend database for the Django app to use, much like Yellowfin's own repository.