Full Stack Exercise

## Initial Setup

Download “acme-app-starter” from GitHub, <https://github.com/bruceEgordon/acme-app-starter> and initialize it.

1. Navigate to <https://github.com/bruceEgordon/acme-app-starter>
2. Select the **Code** drop down menu then click the **Download ZIP** option:  
   Graphical user interface, text, application, email

   Description automatically generated
3. Save the Zip file to a location of your choice, then extract the files (they should extract into a single folder named “acme-app-starter-master”).
4. Open the “acme-app-starter-master” folder in Visual Studio Code.
5. Open a Terminal window in VS Code and run **npm install** to rehydrate the packages.  
   Text

   Description automatically generated
6. From the same terminal window, once the install completes, run **npm run start** to build and launch the app.  
   Text

   Description automatically generated
7. Explore the application as it is currently written and try logging in with a few different names and note that no matter what name you use, you currently always see the button (it expands if you hover over it).  
   Graphical user interface, text, application

   Description automatically generated

* You will use Full Stack in the rest of this exercise to turn the expanding button into a feature flag and control it’s visibility as well as visual aspects.

## Create a new Flag

1. Open a web browser session and log into <https://app.optimizely.com/> with the email you used to sign up for this exercise, or you can use your own account if you have one.
2. If your email is associated with multiple experimentation accounts, you will see a drop-down arrow by your name and you will need to select the **Optimizely Education** account. If you don’t see a drop-down arrow by your name, you only have one account, and you can ignore this step.  
     
   Graphical user interface, text, application

   Description automatically generated
3. Verify you are in a Full Stack project named “Acme Site XX” (the XX will be a number assigned to your login).  
   Graphical user interface, text, application

   Description automatically generated

* If you are using your own account, not one that was provided with the training, you will need to create a new Full Stack project named something like “Acme Site”.

1. In your Full Stack project, in the **Flags** section click the **Create Flag** button.  
   Graphical user interface, text, application

   Description automatically generated
2. Name the new Flag **Expanding Button**, leave the key the default **expanding\_button**,and click the **Create Flag** button.  
   Graphical user interface, text, application, email

   Description automatically generated
3. Click the new **Expanding Button** flag to open it for editing.
4. With the **Development** environment selected, toggle the flag on for the Rules for Development.  
   Graphical user interface, application

   Description automatically generated
5. In the middle **Rules for Development** panel of the new Flag, click the **Add** Rule button and then select **Targeted Delivery**.  
   Graphical user interface, text, application

   Description automatically generated
6. In the **Configure Rule** panel, leave name, key and audiences at their default but move the **Percentage Included** slider to the **50%** mark and change the **Deliver** drop-down to **On** and click the **Save** button.  
   Graphical user interface, text, application

   Description automatically generated

* In a later part of this exercise, you’ll use **Audiences** to filter traffic.

## Install the JavaScript SDK in the demo app

1. Switch to the Visual Studio Code window and in the terminal window type **Ctrl + C** to stop the running application.  
   Graphical user interface, text

   Description automatically generated with medium confidence
2. From the same Code Terminal window run “**npm install @optimizely/optimizely-sdk**” to install the JavaScript SDK.  
   Text

   Description automatically generated
3. From the Code Terminal window run “**npm run start**” to restart the application.

* Note the documentation site for installing the JavaScript SDK as well as the choices of SDK’s, <https://docs.developers.optimizely.com/experimentation/v4.0.0-full-stack/docs/install-sdk-javascript>
* Note the instructions on the option of using a micro service instead of installing the SDK

1. From the Visual Studio Code explorer panel, expand the **src** folder and then open the **FsFeature.js** React component file.  
   A screenshot of a computer

   Description automatically generated with medium confidence
2. Edit the FsFeature.js file and add the necessary “imports” to the top:

import { createInstance } from "@optimizely/optimizely-sdk";

It should look like the following:  
Text

Description automatically generated

1. Edit the FsFeature.js file’s **useEffect** function block by adding the following line:

const optiClientInst = createInstance({sdkKey: 'change key'});

It should look like the following:

A screenshot of a computer

Description automatically generated with medium confidence

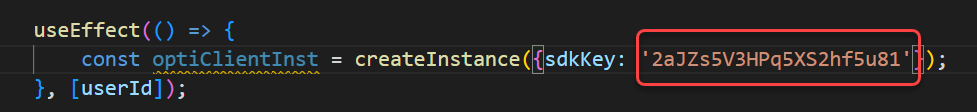
* <https://docs.developers.optimizely.com/experimentation/v4.0.0-full-stack/docs/initialize-sdk-javascript> , scroll down to see more examples and options.

1. Switch to the web browser Full Stack project portal and click the settings cog icon in the main navigation.  
   Graphical user interface, application

   Description automatically generated
2. From the **Settings** page, click the clipboard icon next to the key for the development environment to copy the key into your clipboard (make sure it’s the development environment).  
   Graphical user interface, text, application, email, Teams

   Description automatically generated

* You can click the link to the JSON file if you want to see what’s in the configuration file that gets downloaded by the SDK in the next step.

1. Switch to visual studio code and past the key into the line added previously where it says **change key**, should look like the following but with a different key:  
   
2. Add the following to handle when the promise object returns the initialized client:

        optiClientInst.onReady().then(() => {

        });

It should look like the following:  
Text

Description automatically generated

* The object returned from **createInstance** for JavaScript provides a promise style callback function so you can safely add code that won’t run until the configuration file has been successfully downloaded.

1. Inside the **onRead().then()** function from the previous step, add the following to create user context:

 const user = optiClientInst.createUserContext(userId);

 setTheUser(user);

It should look like the following:  
A screenshot of a computer

Description automatically generated with medium confidence

* The **setTheUser** function is specific to React for saving the user into a state object so that it can be retrieved elsewhere in code.
* The **createUserContext** requires you to minimally pass in a user ID, you carn read more about otptions from the online documentation, <https://docs.developers.optimizely.com/experimentation/v4.0.0-full-stack/docs/create-user-context-javascript> .

1. Below the last line from the previous step, still inside the promise callback, add the following code to get the decision about the user and the feature:

setDecision(user.decide('<flag key>'));

It should look like the following:  
Text

Description automatically generated

* The **setDecision** function is specific to React for saving the decision into a state object so that it can be retrieved elsewhere in code.

1. Switch to the web browser Full Stack project portal and click the **Flags** link from the main navigation menu.
2. Double-click the key for the **Expanding Button** flag to highlight it and then use **Ctrl + C** to copy it into your clipboard.  
   Graphical user interface, text, application, email

   Description automatically generated
3. Switch back to Visual Studio Code and paste flag key from the project UI into the **decide** method call over the **<flag key>** place holder. It should look like the following:  
   Text

   Description automatically generated

* This is where the “bucketing” occurs and where, if it was an experiment, an event would fire and send back the decision results to the Optimizely cloud for the user.

1. Modify the JSX return by adding a line to show the feature decision and button based on the decision:

            {decision && <div>Feature enabled: {decision.enabled.toString()}</div>}

            {decision && decision.enabled &&

                <AnimatedButton handleButonClick={handleButonClick}

                    scaleAmount={2}

                    textColor="black"

                    backColor="lightgray" />}

It should look like the following:  
Text

Description automatically generated

* This wraps the button component in conditional logic that first checks to see if the **decision** variable is **null** and if it is not **null** it then checks the **enabled** property that is the result of decision for the user. The extra line above it is just for displaying the decision result.

1. Save the **FsFeature.js** file changes in Visual Studio Code.
2. Switch the web browser and the Acme site and try logging in with different user id’s. Some users should see the button, and some won’t.  
   The following is what a user that gets bucked into the range that gets the feature flag:  
   Graphical user interface, text, application, email

   Description automatically generated  
   This is what a user would see if they gets bucked into the range that doesn’t get the feature:  
   Graphical user interface, text, application, email

   Description automatically generated

* The online documentation has more on the **Core concepts – How bucketing works**, <https://docs.developers.optimizely.com/experimentation/v4.0.0-full-stack/docs/how-bucketing-works>
* Optimizely uses a hash function that takes the user id and generates a number between 1 and 10,000. This number is compared against the traffic scale to determine if the user gets a feature or a specific variation. The hash function always calculates the same number with the same user id as input, so the bucketing is naturally sticky.

## Create variables

1. In the UI portal for the project Flag’s list, select the **Expanding Button** flag if it is not already selected and then click **Default Variables**.  
   Graphical user interface, application

   Description automatically generated
2. In the default variables panel click the “plus” icon and choose **Integer**.  
   Graphical user interface, text, application

   Description automatically generated
3. In the **Configure Integer Variable** panel, set the **Variable Key** to **button\_scale** and the **Default Value** to **2**.Graphical user interface, text, application, email

   Description automatically generated

* Note the **Code Sample** lets you choose the SDK and shows a code snippet of how to access the variable value.

1. Click **Save** to save the new variable
2. Repeat the steps to create two additional **String** variables, **text\_color** and **back\_color**. Set the default values for **text\_color** to **black** and **back\_color** to **lightgray**. The variables list should look like the following:Graphical user interface, application

   Description automatically generated
3. Switch to Visual Studio Code and in the **FsFeature.js** file, change the **AnimatedButton** element’s **scaleAmount, txtColor** and **backColor** attributes to:

                <AnimatedButton handleButonClick={handleButonClick}

                    scaleAmount={decision.variables.button\_scale}

                    textColor={decision.variables.text\_color}

                    backColor={decision.variables.back\_color} />

1. Modify the information line above the **AnimantedButton** to add the scale value to the output so it’s clear when the JSON configuration file has updated:

{decision && <div>Feature enabled: {decision.enabled.toString()} and scale is: {decision.variables.button\_scale}</div>}

The final version should look like the following:  
Text

Description automatically generated

1. Use **File | Save** to save the changes to the **FsFeature.js** file.
2. Switch to the web browser and make sure the Acme site is working correctly. At this point it shouldn’t look much different except for the added scale value.  
   Graphical user interface, text, application, email

   Description automatically generated
3. Switch to the web browser tab of the UI portal for the **Expanding Button** flag and change the variable **button\_scale** default value to **4** and click the **Save** button.  
   Graphical user interface, text, application

   Description automatically generated
4. Switch to the Acme site browser tab and hold down the **Ctrl** key and click the refresh icon until you see the scale value change to **4**, it may take a few refreshes before it reads the new variable value.  
   Graphical user interface, text, application

   Description automatically generated
5. When you move your mouse over the button it should reflect the larger scale when it animates.  
   Graphical user interface, text, application, email

   Description automatically generated

## Create variations

1. Switch to the web browser tab of the UI portal and in the project’s Flag list, select the **Expanding Button** flag if it is not already selected and then click **Variations**.  
   Graphical user interface, application

   Description automatically generated
2. In the middle **Variations** panel click the **plus** icon.  
   Graphical user interface, application

   Description automatically generated
3. Name it “**Scale of 3**” and set the **back\_color** to **green**, **button\_scale** to **3**, **text\_color** to **white** and click **Save**.Graphical user interface, text, application, email

   Description automatically generated
4. Repeat the previous steps to create another variation and name it “**Scale of 5**” with the **back\_color** to **red**, **button\_scale** to **5**, **text\_color** to **white** and click **Save**.   
   Graphical user interface, application

   Description automatically generated

## Create and experiment

1. In the UI portal for the **Expanding Button** flag, click the **Development** environment and then click the **Add Rule** button from the middle panel and select **A/B Test** from the dropdown.  
   Graphical user interface, text, application, email

   Description automatically generated
2. Leave the percentage included at 100% but change the **Deliver Variations** to the two variations created earlier:  
   Graphical user interface, application

   Description automatically generated

* the “plus” icon next to the last variation lets you add more variations to the experiment.

1. Try to click **Save** and note how you must have at least one metric created for an experiment.  
   Graphical user interface

   Description automatically generated
2. Click inside the **Metric** field and choose the **Create new event** option.
3. Name the event key **scaleButtonClicked** and note the sample code then click the **Create Event** button:

Graphical user interface, text, application, email

Description automatically generated

1. On the **Add Metric** form, select **total conversions** from the dropdown and then click **Add Metric** button.  
   Graphical user interface, text, application, email

   Description automatically generated
2. Click the **Save** button to save the new Experiment.
3. Switch to Visual Studio code and in the **FsFeature.js** file edit the **handleButtonClick** event method to track the button click event:

theUser.trackEvent('scaleButtonClicked');

It should look like the following:  
A picture containing graphical user interface

Description automatically generated

1. Use **File | Save** to save the changes to the **FsFeature.js** file.
2. Switch to the web browser tab with the Acme site and try logging in as different users. Every user should see the button since the experiment is set to 100% but each user has a 50% chance to see a different variation.  
   Graphical user interface, text, application

   Description automatically generated
3. Click the button to record an event.
   * **Note**: You can use Chrome’s developer tools and the network monitor to view event activity and see the button clicked event.

## Audiences

* YouTube video on how audiences work, <https://www.youtube.com/watch?v=tJEveVFR4WE>.
* In online documentation, <https://docs.developers.optimizely.com/experimentation/v4.0.0-full-stack/docs/target-audiences>.

1. Switch to Visual Studio Code and edit the **FsFeature.js** file by modifying the **createUserContext** method and adding the following attribute argument:

{isVip: true}

Text

Description automatically generated

* For this demo you are statically adding this attribute value to the user, in a real-world scenario this data would be dynamic and come from some data source you have available with user information you want to use for audience targeting.

1. Use **File | Save** to save the changes to the **FsFeature.js** file.
2. Switch to the project UI portal and click the **Audiences** link and then click the **Attributes** tab.  
   Graphical user interface, application

   Description automatically generated
3. Click the **Create New Attribute** button and set the **Attribute Key** to match what you typed in the code and click the **Save Attribute** button.  
   Graphical user interface, application, email

   Description automatically generated
4. Click the **Saved** tab and then click the **Add an Audience** button.  
   Graphical user interface, application, Teams

   Description automatically generated
5. Name the new audience “**Vip People**” and drag the **isVip** from the Attributes list to the Audience conditions list and configure it as a **Boolean** with a value of **true**.

Graphical user interface, text, application

Description automatically generated

1. Click the **Save Audience** button.
2. Go back to the Flag configuration and select one rules and note how you can new select the **Vip People** audience to replace everyone as a target for the rule, such as the experiment in the image.  
   Graphical user interface, text, application

   Description automatically generated

* If you have multiple audiences, you can combine them with either an **and** or an **or** to meet more complex scenarios.

# Random Stuff

The topics below are not part of the exercise but are included as useful tips.

## Forced Variations

This can be used for testing or any scenario where you want a user to use a specific variation, <https://docs.developers.optimizely.com/experimentation/v4.0.0-full-stack/docs/set-forced-variation-javascript>.

optimizelyClient.setForcedVariation('my\_experiment\_key', 'user\_123', 'some\_variation\_key')

@return boolean true if the user was successfully forced into a variation, false if the experimentKey isn't in the project file or the variationKey isn't in the experiment.

## Troubleshooting and QA

To see what variation a user has been assigned you can add the following code between the current info line and the button:

{decision && <div>Variation Key: {decision['variationKey']}</div>}

**Note:** This shows how to get the variation key the user is currently bucketed into. The documentation has more information here, <https://docs.developers.optimizely.com/experimentation/v4.0.0-full-stack/docs/decide-methods-javascript>.