Paper 4437-2020

Building Web Applications with SAS® Has Never Been So Easy

Matthew Kastin, NORC at the University of Chicago; Nikola Markovic, Boemska

ABSTRACT

Boemska AppFactory helps you complete the “last mile of analytics” by empowering users to innovate with highly tailored productivity apps. When combined with the SAS Viya Platform you can create sophisticated apps from your data and increase access to analytics by enabling more of your organization to harness the power of SAS by deploying modern, intuitive, and relevant apps.

INTRODUCTION

We are going to explore the process of creating a simple web app using ReactJS and SAS Viya inside of Boemska AppFactory. The app you are creating implements a React component called react-image-annotate by WorkAround Online Inc. (wao.ai), which we will integrate with the SAS Viya APIs. The combination of these components will result in a prototype Image Annotation app that can be rolled out to end users to securely collect image annotations across your organization. This app can then be used to improve the quality of your image processing models.

GETTING STARTED

We have designed this application to be presented as a 45-minute hands-on workshop. The technology described in this paper will be split into two areas: the SAS programming that most readers will be familiar with and Javascript (ReactJS) web application development, which may be more of a novelty. In a real-world scenario these two technologies would likely be taken on by different individuals with strengths in the relative area of development. However, in the context of our Hands-on Workshop you will be working solo and wearing both hats, so to speak. Boemska AppFactory also makes assuming whichever role(s) simple.

To complete the app, we will perform the following tasks for each role:

1. Web developer
    i. Install the Javascript libraries required by the seed app
    ii. Build the seed app and test the initial build in the browser
    iii. Create a new page in the app
2. SAS programmer
    i. Create a new project in Boemska AppFactory
    ii. Create a folder and a data service

iii. Write the SAS code which will return the dataset of images to the client
iv. Test and deploy the code
v. Take a copy of the generated React JS snippet from AppFactory

3. Web developer
   i. Add the generated code from AppFactory to the page you last created
   ii. Test that the app is communicating with Viya
   iii. Add the React Image Annotate component to the app
   iv. Add the same component to the page and configure it
   v. Test, and show off to your friends

By the time you are done the app should look like Figure 1:

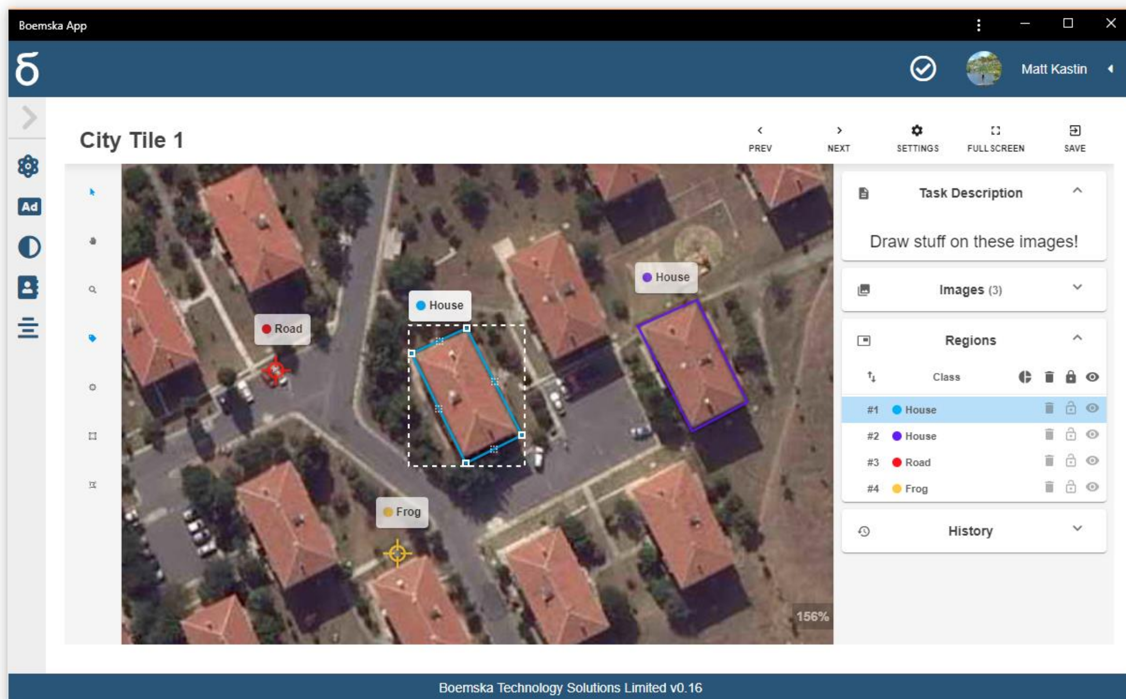

Figure 1. Preview of Final Application

ENTER WEB DEVELOPER MODE

For the workshop, you will be provided your own login to a browser-based instance of Microsoft's VSCode editor. Open the URL provided, and you will be greeted with a screen that looks like Figure 2:

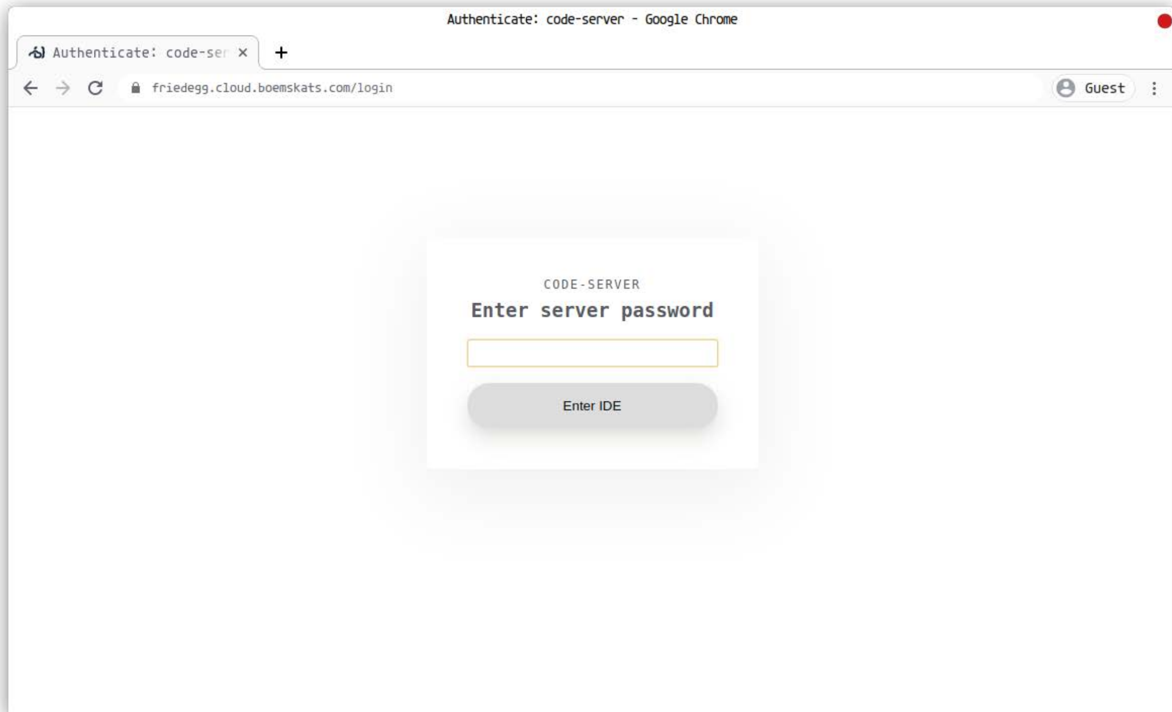

Figure 2. Microsoft VSCode Editor

Once you have logged on successfully, VSCode will provide you with a Terminal instance in your Home directory (shortcut Ctrl-`), where you can use Git to clone the Boemska react-seed-app project into a directory called workshop-app. Snippet 1 shows the command to run.

```
1. git clone git@builds.boemskats.com:boza/react-seed-app workshop-app -b dev-no-eject
```

Snippet 1. Git Clone Command for Boemska react-seed-app

Navigate to the newly create directory, and run `yarn install` to tell the yarn package manager to download the project dependencies to the project. Figure 3 shows what your screen should look like at this point.

```
~/projects/workshop-app

┌[friedegg][a][~/projects]
└▪ git clone git@builds.boemskats.com:boza/react-seed-app workshop-app -b dev-no
-eject
Cloning into 'workshop-app'...
remote: Enumerating objects: 322, done.
remote: Counting objects: 100% (322/322), done.
remote: Compressing objects: 100% (177/177), done.
remote: Total 2059 (delta 203), reused 229 (delta 140), pack-reused 1737
Receiving objects: 100% (2059/2059), 6.49 MiB | 10.08 MiB/s, done.
Resolving deltas: 100% (1249/1249), done.
┌[friedegg][a][~/projects]
└▪ cd workshop-app
┌[friedegg][a][±][dev-no-eject ✓][~/projects/workshop-app]
└▪ yarn install
yarn install v1.22.0
info No lockfile found.
[1/4] Resolving packages...
. h54s@git+ssh://git@builds.boemskats.com/chris/h54s.git#release-candidate
```

Figure 3. Terminal Output from Git Clone and Yarn Install

Once this has completed, run `yarn build` to build the app.

To test the app, navigate to the document root directory you have been provided for your user (it will look something like https://a.cloud.boemskats.com/~username/workshop-app/) and open the build directory from that location. Figure 4 shows what your screen should look like:

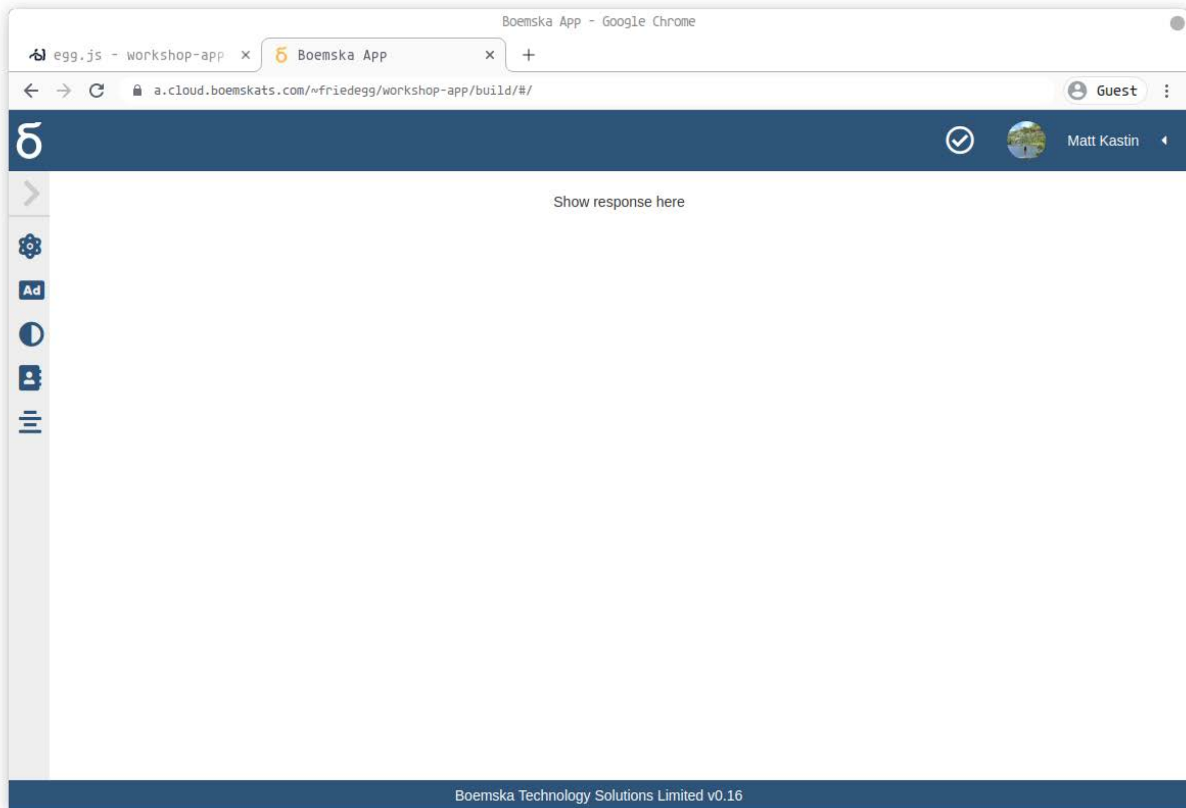

Figure 4. ReactJS Seed Application Preview

This is enough for now. You have taken a copy of the seed app, installed the libraries it depends on, and compiled the app. Next, it's time to put your SAS Programmer hat back on and write some data code.

ENTER SAS PROGRAMMER MODE

For our app to do anything, it needs data - and this is where our SAS code comes in. Next, you will write SAS code that creates a dataset of available image files and feeds it back to the front-end of the application. You do that in AppFactory.

To log on to AppFactory, navigate to the AppFactory URL provided (will look something like http://a.cloud.boemskats.com/apps/), and log on. Select the Projects tab, click Create New, and give your new project a name, description, and a root folder in the Folder Service. It should look like Figure 5:

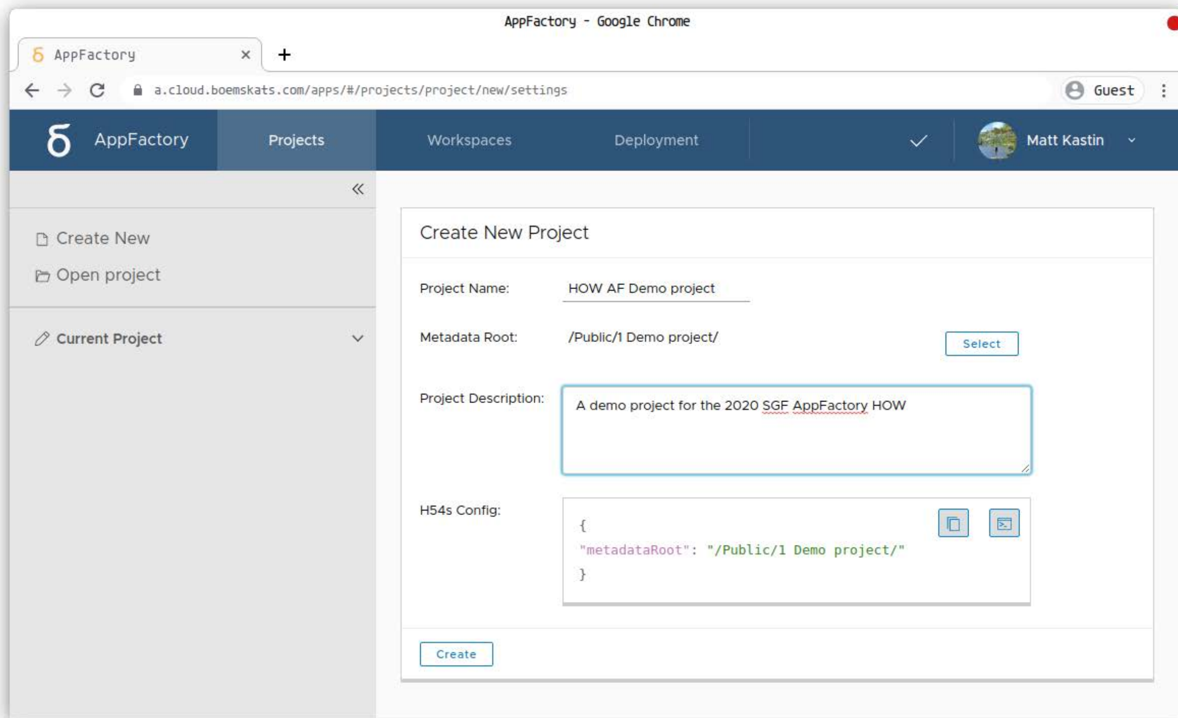

Figure 5. AppFactory: Create New Project Screen

Clicking Create will create the new project in AppFactory, which will allow you to define new roles (folders), services (jobs) within those folders, and write and test the code for each service.

Once your project is created, add a Role. This will be a folder in the folder service, the permissions to which can be mapped to a group of users according to the chosen authorization model. Figure 6 shows an example of the roles screen:

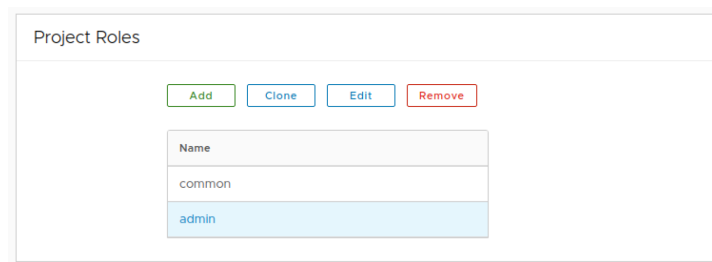

Figure 6. Project Roles

Next, it's time to create a service within one of those roles. Double-clicking a role will open that role. There you will create a SAS service called get image list.

When Roles and Services are created within an AppFactory project, they are not created within SAS until the project elements are explicitly synchronized. Clicking the 'Create' button that appears on the page of any element that doesn't yet exist within SAS, be that a folder or a service, will create that element. You will see a notification like in Figure 7:

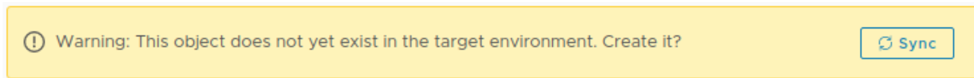

Figure 7. Object does not yet exist notification

Now your 'get image list' service is defined, you need to specify the table metadata for the inputs and outputs that the service will be expecting and providing, before writing the code to process that data and return the results.

You do this by clicking 'Add' in the Data I/O section of the service page. This allows you to then specify the structure of each dataset that will be either expected or produced by your code.

This service will be providing a set of image names and image URLs to the app. So, the data structures it produces will look like Figure 8:

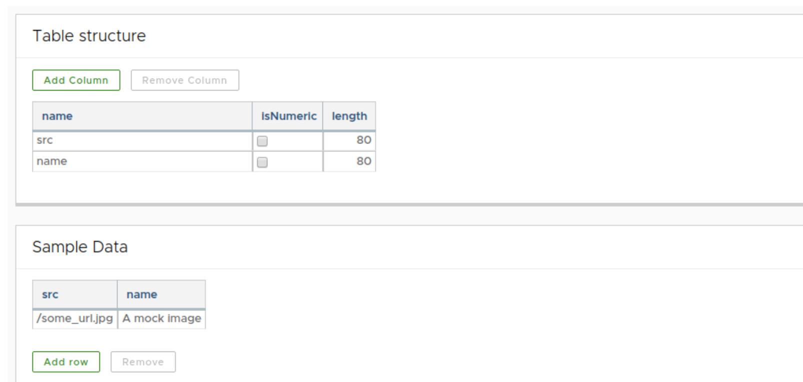

Figure 8. AppFactory: Data I/O Section of the Service Page

Define these, save them, and then click the Edit Code button to write the SAS code that will produce this dataset.

For the workshop, a pre-prepared dataset will be made available which contains a set of image URIs on the Viya Files service, and the associate image names. To return these to the app front end, the SAS code can remain simple. Snippet 1 contains all the SAS code we need and Figure 9 shows it in Boemska AppFactory:

```
1. libname imgrepo '/pub/imagerepo';
2.
3. proc sql;
4.   create table candidates as
5.   select src, name from imgrepo.candidates;
6. quit;
```

Snippet 2. SAS Code for prepared dataset

/Public/1 Demo project/

Test

```
1  /***--------get image list--------***/
2
3  %bafGetDatasets; /* get all input tables */
4  resetline; /* for error reconciliation */
5
6  /*** START USER WRITTEN CODE ***/
7
8
9  libname imgrepo '/pub/imagerepo';
10
11 proc sql;
12   create table candidates as
13   select src, name from imgrepo.candidates;
14 quit;
15
16
17 /*** END USER WRITTEN CODE ***/
18 * if output data does not exist yet use sample data ;
19 %macro bafCheckoutputs;
20 * this if sysfunc exist happens for each outtable ;
21
22 * --out table 0-- ;
23   %if %sysfunc(exist(image_list)) = 0 %then %do;
24     data image_list;
25       length src $80. name $80.;
26       src = '/some_url.jpg'; name = 'A mock image'; output;
27     run;
28   %end;
29 %mend; %bafCheckoutputs;
30 %bafheader;
31     %bafOutDataset(image_list, work, image_list);
32 %bafFooter;
```

Inputs

Outputs

image_list

```
length
   src $80
   name $80
;
```

Figure 9. SAS Code as seen inside AppFactory

Once your code is written, clicking on the Test button in the top right of the code editor will allow you to run it and test that it produces the desired output. Assuming it does, clicking the Save Code button (the rightmost of the four buttons along the top left of the editor window) will save your code changes back to the project, and navigate back to the main page for that service. Clicking the 'Update' button from there will ensure that your code changes are synchronized with the deployed project code.

After this is done, scroll to the bottom of the Service page, and inside the Javascript Code section, select React and click the Copy Code button. Figure 10 shows what the Service page looks like:

Javascript Code

Copy Code

Vanilla JavaScript | Angular | React

```
// adapterService should be your wrapper around h54s
import React from 'react'
import adapterService from './myCustomFolder/myCustomServices/adapterService'

class MyClassName extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      response: null,
      error: ''
    }
  }
  async componentDidMount() {

    try {
      const res = await adapterService.call('common/get image list', null);
      console.log(res);
      this.setState({response: res, error: ''});
    } catch(err) {
      this.setState({error: err.message})
```

Figure 10. AppFactory Service page

Now it’s time to go back to being a Web Developer.

ENTER WEB DEVELOPER MODE (AGAIN)

Logging back onto VSCode in the Browser, you should be able to open your previously opened project as seen below in Figure 11:

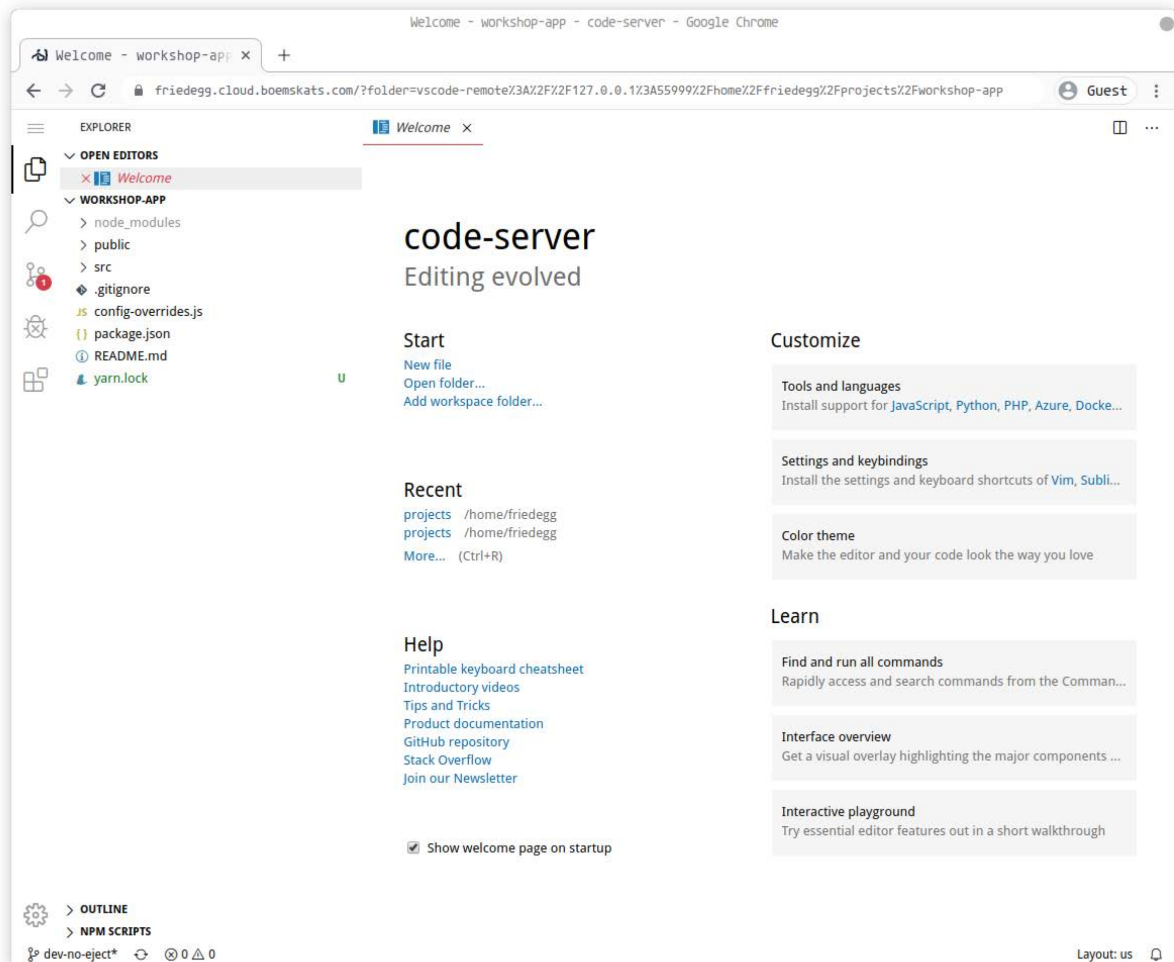

Figure 11. VSCode Project

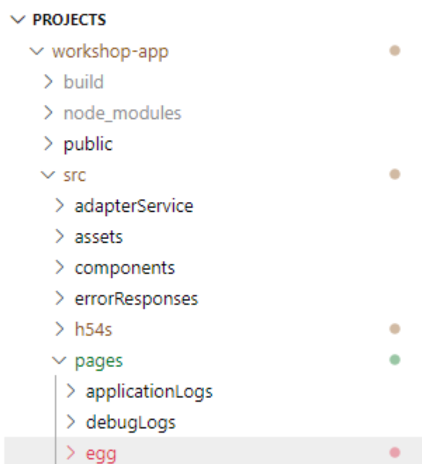

Figure 12. Explorer panel

The next step is to create the page where the react-image-annotate component will be shown. This involves two steps: creating the page itself, and then **adding it's route to the main application's App.js file.**

The first step involves creating a subdirectory inside the project structure, and a new .js file inside that subdirectory. Figure 12 shows the Explorer panel on the left-hand side of VSCode to highlight this action:

Inside the new file, paste the code that AppFactory generated, copied from the previous step. There are a couple of changes that need to be made, to ensure that the adapterService is being sourced from the correct path, and that the new class is named correctly.

In this case, the three key parts of your code should read like lines 1, 3 and 5 of Snippet 3:

```
1. import adapterService from '../../adapterService/adapterService'
2.
3. class Egg extends React.Component {
4.
5. export default connect(mapStateToProps, mapDispatchToProps)(Egg)
```

Snippet 3. React component update excerpts

Then, adding this new page to the application requires the following two changes to App.js highlighted below in Figure 13:

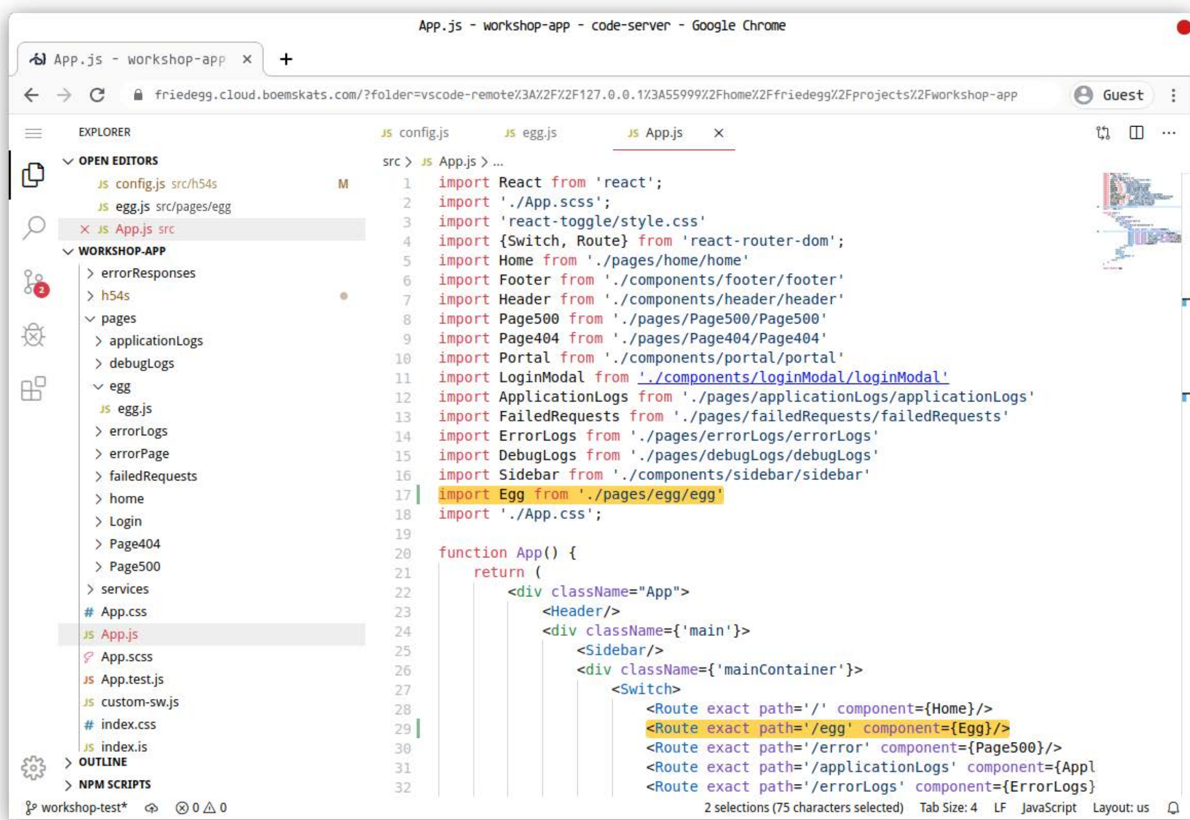

Figure 13. App.js Highlighted Changed

The last step before the next test is a change that needs to be made to the default h54s adapter configuration, to tell it where the app's services have been deployed within the Viya folder service. This is done by editing the h54s/config.js file, as shown in Figure 14:

workshop-app > src > h54s > JS config.js > ...

```
1  export default {
2      metadataRoot: "/Public/1 Demo project/",
3      hostUrl: '',
4      ajaxTimeout: 120000,
5      sasVersion: 'viya' // could be 'v9' or 'viya'
6  }
```

Figure 14. H54s config.js example

Note that this config can be copied directly from the main Project page in AppFactory, like in Figure 15:

H54s Config:

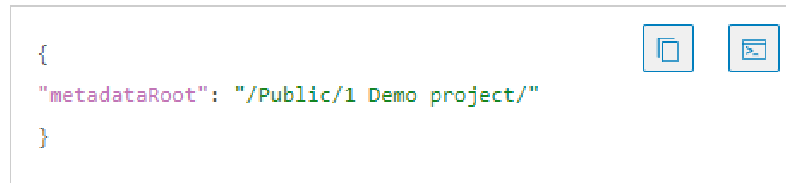

Figure 15. Alternative H54s config example inside AppFactory

Once this change is made, it is time to build the app once again and verify that loading our new page successfully retrieves the data it needs from SAS. To do this, run the `yarn build` command in the VSCode terminal console again, and then load the app back up in the browser - this time appending an `/egg` uri to the address of the app to point at the new component.

This time, before hitting the enter key, open the browser's Developer Tools console. By default, the snippet of React code copied from AppFactory contains a console.log(response) command, which you will substitute in the next step. For now, it can show us that the data from Viya is being retrieved correctly. Therefore, loading the URL with the new page appended to it should yield a response like this in the browser console (see Figure 16):

```
                                                                                    egg.js:20
{image_list: Array(3), usermessage: "blank", logmessage: "blank", requestingUser: "friedegg",
requestingPerson: "friedegg", …}
 ▼image_list: Array(3)
   ▶0: {src: "/files/files/2a1265c9-37ac-4cc3-8790-f3a0d9bde91d/content", name: "City Tile 1…
   ▶1: {src: "/files/files/df3610c8-0c5e-4c1d-9509-cc0abc902b9c/content", name: "City Image"}
   ▶2: {src: "/files/files/61629bdc-28a6-4dfa-910c-d0a5c74b7e83/content", name: "Road 2"}
    length: 3
   ▶__proto__: Array(0)
  usermessage: "blank"
  logmessage: "blank"
  requestingUser: "friedegg"
  requestingPerson: "friedegg"
  executingPid: 10350
  sasDatetime: 1899130167.7
  status: "success"
 ▶__proto__: Object
```

Figure 16. Web browser developer console

This shows the successful response on the load of that page, including the image list data. Great. Time to move onto the next and final step - adding the react-image-annotate component to our project.

This is done using the yarn package manager, back in VSCode's Terminal. It involves typing `yarn add react-image-annotate`, and should yield something like in Figure 17:

```
┌─[friedegg][a][±][workshop-test U:2 ?:2 ✗][~/projects/workshop-app]
└─▪ yarn add react-image-annotate
yarn add v1.22.0
[1/4] Resolving packages...
warning react-image-annotate > transformation-matrix-js@2.7.6: Package no longer supported
. Contact support@npmjs.com for more info.
warning react-image-annotate > @material-ui/core > popper.js@1.16.1: You can find the new
Popper v2 at @popperjs/core, this package is dedicated to the legacy v1
warning react-image-annotate > material-survey > react-dropzone > attr-accept > core-js@2.
6.11: core-js@<3 is no longer maintained and not recommended for usage due to the number o
f issues. Please, upgrade your dependencies to the actual version of core-js@3.
warning react-image-annotate > react-json-view > flux > fbjs > core-js@1.2.7: core-js@<3 i
s no longer maintained and not recommended for usage due to the number of issues. Please,
upgrade your dependencies to the actual version of core-js@3.
[2/4] Fetching packages...
[##########################################################################-----] 1633/1737
```

Figure 17. Adding react-image-annotate

Once this action has completed, the react-image-annotate component will be available within your project. Next, it is a simple case of following the instructions from the component's homepage (https://waoai.github.io/react-image-annotate/). Namely, it involves importing the component into the newly created page, by importing it. Snippet 4 highlights the code to do this:

```
1. import ReactImageAnnotate from "react-image-annotate"
```

Snippet 4. Importing react-image-annotate

and then configuring it, as show in Figure 18:

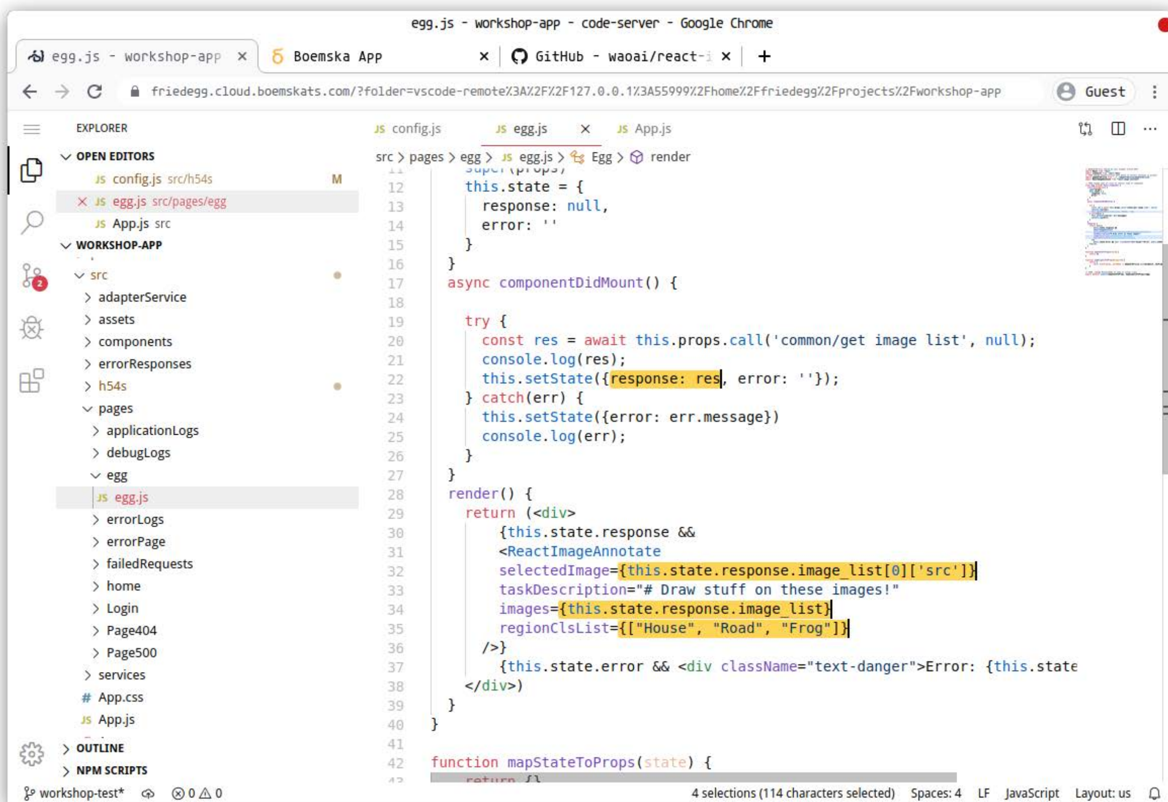

Figure 18. Configuring react-image-annotate

These changes do the following:

- Set the response object to update the global state of the application when it returns
- Tell the ReactImageAnnotate component to use the image list object from the response as its dataset listing all the eligible source images
- Tell it to use the first ([0th]) element in the array as the selected image
- Tell it to also offer three categories for object classification. These could also be made data driven easily by producing a second dataset from our AppFactory service that lists some more interesting categories

And that's it. Running `yarn build` one last time should build a version of the application, and hitting refresh should result in an image annotation interface show in Figure 19:

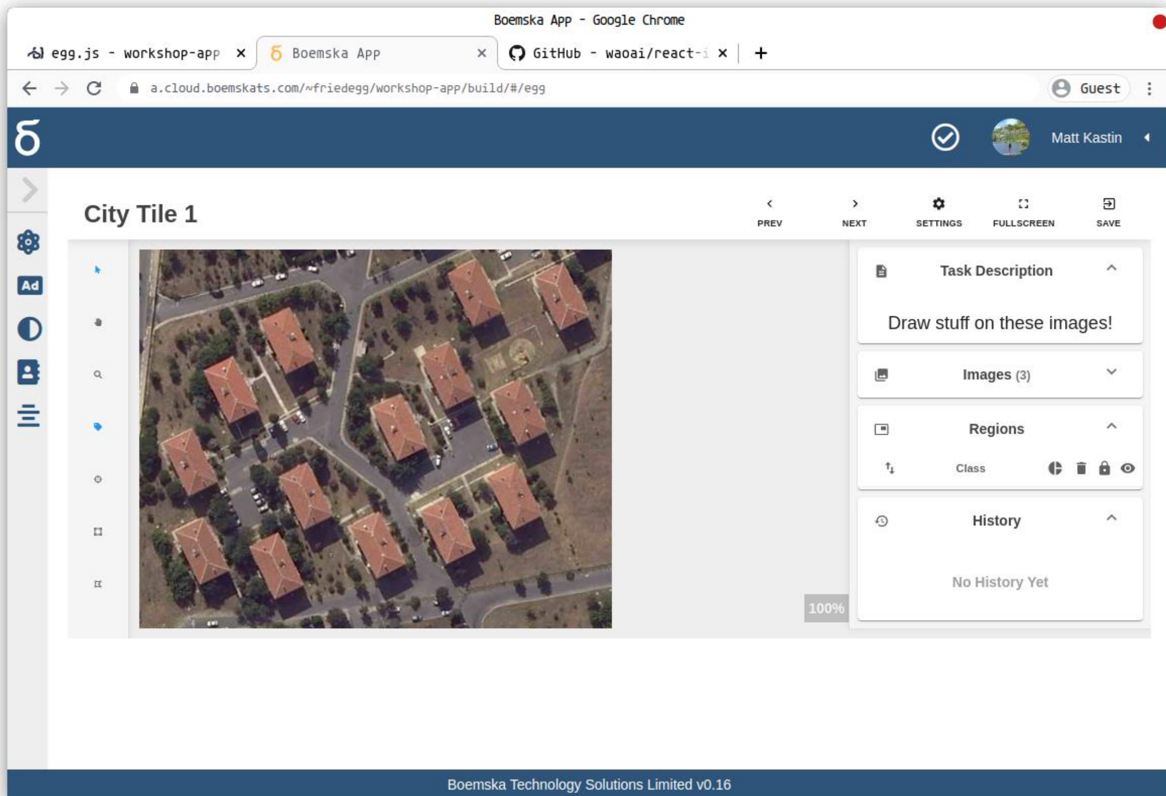

Figure 19. Tadaa!

The next obvious step is the development of this app is to create and implement a service that writes the resulting annotations back to an audited dataset. While that's outside the scope of this workshop, a continuation of these materials should soon be available online. Watch this space!

CONCLUSION

Congratulations! You have completed building a secure, image intake web application for pre-processing and storing images and metadata. This example app demonstrates how Boemska AppFactory helps fill in the gaps in the last mile of analytics. Reducing the complexity on both sides of development by following a structured approach.

From here, your imagination is truly your only limit (when it comes to building web applications to expand the capabilities of SAS)!

RECOMMENDED READING

- https://boemskats.com
- https://reactjs.org/

- https://github.com/boemska/h54s
- https://waoai.github.io/react-image-annotate

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Matthew Kastin
NORC at the University of Chicago
fried.egg@verizon.net

Nikola Markovic
Boemska
nik@boemskats.com
http://boemskats.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.