Ecosystem-based Automated Range Mapping (EBAR) Reviewer Web App Technical Document

Prepared for: Nature Serve Canada

By: Kevin Hibma, Esri Canada

January 2020

Contents

[Introduction 2](#_Toc29795024)

[Setting up a development environment, building and debugging 2](#_Toc29795025)

[Modules / Dependencies 4](#_Toc29795026)

[Code 4](#_Toc29795027)

[Deploying 5](#_Toc29795028)

# Introduction

The Ecosystem-based Automated Range Mapping (EBAR) Reviewer is a web application that makes use of a feature service and basemaps, drawing species range information in a map, where experts can review these areas and provide feedback.

The web application was built with JavaScript, primarily making use of WebPack and Node.js to compile and bundle the application from both core and Esri specific modules. The application requires node.js to run locally and build, while a compiled version of the application can be served from almost any modern webserver (eg. IIS).

Live application: [https://gis.natureserve.ca/ebar/#](about:blank)

Source Code (github): [https://github.com/NatureServe-Canada/EBARReviewer](about:blank)

Services required by the app are those in the EBAR Reviewer group - <https://gis.natureserve.ca/arcgis/home/group.html?id=301e5d0a64774e68b7fb64d5bf9b8f58&view=list&start=1&num=20#content> (requires user account).

Historical:

This app is based on an application built by Esri Inc for the MoBI project: [https://github.com/vannizhang/species-reviewer](about:blank)

That initial application was enhanced by NatureServe for the NABA project: [https://github.com/allenanselmo/species-reviewer-naba](about:blank)

Range Map inputs:

EBAR Reviewer relies on the EBAR-KBA geodatabase hosted on NatureServe Canada’s ArcGIS Enterprise/Server. This system includes tables and geoprocessing services for importing species observations and generating range maps from them. For additional geodatabase and geoprocessing documentation see https://1drv.ms/w/s!Ajv6BHSXrqqqmc5bm6NNAoTdVxjPJQ?e=vQ9NIR.

User Guide:

User-oriented documentation for EBAR Reviewer is available at <https://1drv.ms/w/s!Ajv6BHSXrqqqmch1OmiznS5QdeintQ?e=Mip1AG>.

# Setting up a development environment, building and debugging

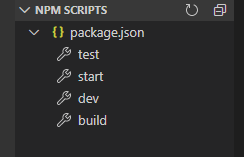
1. Install Node.js: [https://nodejs.org/en/download/](about:blank)
2. Install an IDE to edit code. Visual Studio Code is a free, powerful but lightweight editor that works well.
3. If using VS Code, add the following extensions:

* Debugger for Chrome
* ESLint
* NPM

1. Clone (download) source code: [https://github.com/NatureServe-Canada/EBARReviewer.git](about:blank)
2. Install project dependencies (packages) from command line: npm install
3. Run the application by from command line: npm run start This will open a browser pointing to [http://localhost:8080](about:blank) with the application

Optional build/run/test which may make debugging easier – These instructions assume using VS Code, with the previously mentioned extensions and Google Chrome.

1. Set your break points as needed
2. From the NPM Scripts in VS Code: **Build** the application\*
3. From the NPM Scripts in VS Code: **Start** the application



1. Close the window that was automatically opened from the Start command (It is not hooked up to the debugger)
2. Ensure you have a **/.vscode/launch.json** file configured properly. This file is included in the github repro. Otherwise add/update this config:

        {

            //https://github.com/Microsoft/vscode-chrome-debug/issues/542

            "type": "chrome",

            "request": "launch",

            "name": "Launch Chrome against localhost",

            "url": "http://localhost:8080",

            "webRoot": "${workspaceFolder}/dist",

            "sourceMaps": true,

            "disableNetworkCache": true,

            "sourceMapPathOverrides": {

                "webpack:///\*": "${webRoot}/../\*",

                "webpack:///./\*": "${webRoot}/../\*",

                "webpack:///src/\*": "${webRoot}/../src/\*",

                "webpack:///./~/\*": "${webRoot}/../node\_modules/\*"

            }

        }],

1. Launch the application from the debugger (F5)
2. The new Chrome window that opens will be hooked to the debugger and hit break points

\*When developing/debugging the application will sometimes crash when hitting a certain break point. This is almost always because the breakpoints within VS Code have got out of sync with the source maps > bundle files. Sometimes closing and re-opening VS Code is enough to fix. Alternatively, deleting the **/dist** folder, running a **build** and then **start** will always fix/sync break points

# Modules / Dependencies

The application has several development package dependencies which are handled through Node and NPM to facilitate the build process. The application only has 3 dependencies for its general use:

* Calcite-web: HTML and CSS styles developed by Esri
* Esri-loader: A smaller library to load Esri JavaScript API modules as needed
* Jquery: A common JavaScript library with tons of helpers

# Code

The coding pattern of the EBAR application follows a MVVM\* pattern. That is, code has been segregated into areas of UI, and a data model with a controller gluing these two pieces together.

*This discussion is centered around the development of the application, NOT the deployed application.*

The main entry point to the application is **index.js** with a **config.js** file supplying the application URLs, field mapping, symbology and various other items.

The **index.js** file is responsible for the initial app load and calling:

* an **OauthManager** for authentication
* a **MapController** to handle drawing the map and graphics
* a **View** to handle the generation of UI components
* a **Controller** which works with the data model (explained more below)
* a couple of other supporting pieces

*Controller* is responsible for loading upthe following:

* **DataModel** – a high level object that is populated on app load with species for the logged in suer
* **FeedbackDataModel** – a handler for the **FeedbackManager** that gets and submits feedback
* **ApiManager** – to request (query) and post feedback (updateFeatures, addFeatures) to the feature service.

The **DataModel** and the **FeedbackDataModel** work hand-in-hand when submitting and getting feedback. Note that the *ecoAtts object* variable within the FeedbackModel is not actually needed as part of the model for feedback submission purposes. This was added to the Model as a way to populate the FeedBack UI with information about the selected EcoShape. The attributes inside this object are static from the features and do not get posted in any feedback submission.

\*This is my take on the development pattern. Someone else may disagree and say its more of a MVC pattern. Regardless of MVC or MVVM, the code has been split to generate UI and a data model, with components that allow the two pieces to talk.

# Deploying

Follow the steps in the **Setting up a development environment, building and debugging** section to build the application. Before deploying, the **/dist** folder should be deleted, and a fresh build process should be started.

Copy the entire contents of the /dist folder to a directory within your given web server.

The application is now available from its given URL, specifically from the *index.html* start point.

**Note –** To deploy an application from a local development machine to the gis.natureserve.ca Azure VM, you will generally need to ZIP the contents of the local /dist folder and copy the zip to the VM. As the c:\inetpub\wwwroot folder on the VM requires elevated privileges to write to, you’ll need to extract the contents of the ZIP file to a user accessible location on the VM, and then delete the c:\inetput\wwwroot\ebar contents and copy in the new contents (agreeing to the admin warnings).