S.A.R.C.I

Search And Rescue Coastal Intelligence

User Manual

5/04/2022

Team Mentor: Han Peng

Group Members: Dylan Woolley, Vidal Martinez,
Randy Duerinck, Jabril Gray

Team Sponsor: General Dynamics Mission Systems

Table of Contents

Introduction

Thank you for choosing the Rescue21 SWAPR Data Dashboard as an addition to the Rescue21 system! The SWAPR Data Dashboard is a powerful tool that will serve as an additional monitoring tool for the Rescue21 system. The tool will collect, store, analyze, and display weather and power data coming from the SWAPR device through a secure graphical web interface. Some of the key features include: Account Authentication with Role-based Permissions, Summary List View of Weather Data, Summary Map View of Location & Status, and Historical View of Averaged Data through line, bar, and radar graphs, two different notification views showing critical events and managing them, and csv data exporting functionality. The purpose of this user manual is to assist you, the client, successfully install, administer, and maintain the Rescue21 SWAPR Data Dashboard product in your business context going forward. Our hope is to make sure that you can profit from our product as much as possible for many years to come!

Installation

As part of final delivery, the SWAPR Data Dashboard system should have been installed on Windows (For local testing) and Linux (For AWS hosting). Over time, however, you may want to move to a new platform or re-install the product in your own infrastructure. Throughout this next section, we will give the general step-by-step instructions to set up our project on the new system.

The first step when moving the product to a new platform is to install all the .NET dependencies which includes .NET architecture and Visual Studios (if development will take place on the platform). You can download the .NET architecture and Visual Studios at https://visualstudio.microsoft.com/downloads/. If you are only looking for the .NET architecture then you will go to the “Other Tools, Frameworks, and Redistributables” and select the Microsoft Visual C++Redistributable for Visual Studios 2022 and/or .NET Core and .NET Framework.

There is one piece of independent software that needs to be downloaded if the project will be simulated in a lab environment. That software is called the Com0Com which will be responsible for creating virtualized com ports for the Simulator to communicate with the Reader. This software can be downloaded from http://com0com.sourceforge.net/ by clicking the “SourceForge” button at the bottom of the page.

Once you have the architecture and dependencies installed, you will need to pull the code for our GitHub repository. The GitHub repository was added to Amir Vahidi’s one drive account. Once you do find the code repository, you will need to download it or clone it. We will assume that you are downloading the code. To download the code, you will click the green “Code” button on the top right side corner of the main part of the page. You will then click “Download ZIP”. This will download a zip file of the repository content. Once it finishes downloading, extract it to your directory of choice. You now have all of the code for the project.

Now that the code has been downloaded, we can start looking at setting up each subsystem. Let’s start with the Simulator and Reader subsystems. The Simulator is going to pretend to be a SWAPR device at some RFF Tower. In order for the Simulator to work properly, it will need a SiteId to replicate and the com port to communicate over (Com port can be virtualized). If you decide to use the Simulator then you are also creating virtualized com ports. The reader on the other hand, needs to know the com port to communicate on, the aws access key id, the aws secret access key, and the aws sqs queue url. These configuration details can be set in the App.config file and / or passed through the command line. This will allow you to have the Reader setup for a real SWAPR device or for lab environment testing using the Orchestra.bat file to setup the SWAPR Network.

In order to use the Orchestra subsystem to create a SWAPR Network, you will need to navigate to the project folder. The project folder will contain a file called “Orchestra.bat”. This file is what you will use to automatically create the virtualized com port and create the Simulator /

Reader pairs. Before you use the Orchestra, you will need to configure it. This can be done by right clicking on the file and editing it with whatever text editor you like most. There are several configuration options available. You can configure the starting sending and receiving ports (Default: Com3 (3) & Com4 (4)) as well as the number of pairs to create (the number of sites to have) and the site id to start on (configured on call through command line). Once you have configured these details, the Orchestra is ready to run.

In order for the Simulator / Reader pair to work, you will need an AWS environment setup. We have already set up an environment that works for GDMS on AWS so we will briefly explain the process to give an overview of what we did should it need to be replicated later. We are currently using four services directly to get the data from the SWAPR devices to the database. Those services are AWS Message Queuing Service (AWS SQS), AWS Simple Notification Service (AWS SNS), AWS Lambda Functions, and AWS Relational Database Services (AWS RDS). AWS SQS is where the Reader sends its data off to. This service will queue the messages and notify AWS SNS that new messages are in the queue. AWS SNS will then tell AWS Lambda Function that we have messages in the queue. AWS Lambda Functions will then retrieve the data from the queue and process it creating Entries and Notifications that are stored in the database (AWS RDS). To get these services to work together, you will need to create an instance of each. You will then connect the AWS SNS with the AWS SQS by giving permissions on both sides of each service. You will then set up the Lambda Function to take data from the AWS SQS queue. You will need to give the appropriate permissions to the Lambda Function in order to do this operation. The Lambda Function will need to have the project folder named SQSLambdaFunction uploaded to the service with the updated Database information in it. This will then connect to the database and create new entries. The way that you connect the SQSLambdaFunction with the Database is using a connection string that will allow access. This can be retrieved during setup of the database. Don't forget to update the database connection string on line 64 in Function.cs before using the SQSLambdaFunction. Once all of this has been completed, you will have the data generation and collection problem solved. That means you can run the Simulator and Reader pair as much as you'd like.

The next part of the system that needs to be set up is the data analysis side of the project. The only thing that needs to be done for this section is to start the website. This can be done locally through visual studios or online using AWS. If done locally then it will only need to be run with the updated database connection information. This can be updated using the Appsettings.Development.json file. Once the DatabaseContext has been updated, the website should be ready to run locally. If you want to run the website on AWS then you will need to use the AWS Elastic Beanstalk service. The AWS Elastic Beanstalk service will easily setup the project website just using the code. This can be done in the AWS interface or the AWS Tools in Visual Studios. I personally use AWS Tools in Visual Studios (requires programming service account credentials to be inputted into Visual Studios). However, it is easy to also upload the zip code directly to AWS in their web interface. You can learn more about this process at https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.deploy-existing-version.html.

Configuration and Daily Operations

Now that you have the project installed and working, you will need to configure a few things to get it working just right. We have already mentioned some of the configuration options available but we will now list the configuration options by subsystem describing exactly what each one does.

- Simulator
    - Id
        - Site Id that Simulator is generating data for
    - Port
        - Virtualized Com Port to communicate over
- Reader
    - Port
        - Virtualized Com Port to communicate over
    - awsAccessKeyId
        - The Access Key Id for the service account credentials used AWS SQS
    - awsSecretAccessKey
        - The Secret Access Key for the service account credentials used for AWS SQS
    - queueUrl
        - The AWS SQS queue url that data is sent to
- Orchestra
    - Id
        - The Site Id to start generating data on (auto-incrementing by 1)
    - numOfPairs
        - Number of RFF Sites to Simulate
    - firstPort
        - First port for Simulator (Default: Com3)
    - nextPort
        - First port for Reader (Default: Com4)
    - Line 13: Directory to com0com
    - Line 32: Directory To Simulator.exe
    - Line 35: Directory To Reader.exe
    - Line 48: Directory To Root Project
- SQSLambdaFunction
    - Line 64 in Function.cs: Database Connection String
- Website (Only Custom Config Settings Discussed)
    - ConnectionStrings:DatabaseContext
        - Connection String for the database containing entry and notification data
    - ConnectionStrings:IdentityDatabaseContext
        - Connection String for the database containing user account and role data
    - MainSettings:ListViewConfig
        - widgetWidth

        - Total widget's width
    - widgetHeight
        - Total widget's height
    - siteNameYPadding
        - Y padding for the site name
    - compassRadius
        - Radius of the inner cardinal compass
    - compassArrowSideLength
        - The side length of the arrows to be created
    - cardinalXPadding
        - Distance the cardinal direction letter is away from compass on X
    - cardinalYPadding
        - Distance the cardinal direction letter is away from compass on Y
    - cardinalLetterXPadding
        - Used to center the cardinal letters on the compass on X
    - cardinalLetterYPadding
        - Used to center the cardinal letters on the compass on Y
    - widgetXPadding
        - The x padding between widgets
    - widgetYPadding
        - The y padding between widgets
    - widgetsPerPage
        - number of widgets to display on a page
    - widgetsPerRow
        - number of widgets to display on a row
- MainSettings:MapViewConfig
    - maxLatitude
        - The max latitude for the map image used (real world coordinates)
    - minLatitude
        - The min latitude for the map image used (real world coordinates)
    - maxLongitude
        - the max longitude for the map image used (real world coordinates)
    - minLongitude
        - the min longitude for the map image used (real world coordinates)
    - mapWidth
        - Longitude pixel count
    - mapHeight
        - Latitude pixel count
    - markerRadius
        - the radius of the map markers and area tags
    - longitudePadding
        - the padding on the longitude for the markers displayed on the map
    - latitudePadding
        - the padding on the latitude for the markers displayed on the map

        - mapFolder
            - The folder containing the map data
        - sourceMapFileName
            - The name of the source image to use
        - resultMapFileName
            - The name of the result image to use
    - MainSettings:IdentityConfig
        - ForcedAdminAccount
            - The account that will be forced into the Administrator role
        - ForcedUserAccount
            - The account that will be forced into the User role
    - MainSettings:CSVConfig
        - csvFolder
            - Folder to contain all csv data
        - ResultCSVFileName
            - File name to put the exported data into & name of file that is downloaded by the user

By this point in the manual, you should have all of the subsystems setup and configured ready to be used. You should be able to use the project now. However, when running the project you will notice that everything is locked. You will need to sign in to get the product to work. If you are using the environment that we setup then you can use these two accounts:

- Account: admin@gd-ms.com
    - Password: Password1!
- Account: user@gd-ms.com
    - Password: Password1!

Otherwise, you will need to set up two accounts to use. You can do this by setting the MainSettings:IdentityConfig ForcedAdminAccount & ForcedUserAccount. Next, you will need to start the website and register the two accounts names set. So you would click register and use the email address given in the MainSettings:IdentityConfig ForcedAdminAccount & ForcedUserAccount. This will bring you to a new page asking you to confirm your email. Click confirm. This will then create the account but it will not be set to any role. (Note: to do this process only once create the user account first then the admin account). Once both accounts were created, sign in as the admin user and access the Administrator page. This page will force these accounts into their respective roles. The other option is for you to push accounts into the database which will do the same thing. Note: if the Administrator page will not display then add the @Body tag underneath line 56 in Shared/MainLayout.razor. This will change it so that the body is shown to users that are not signed in. You can then access the page to force the account into the database and then remove that line to lock the project again.

Now that you have account access to the website, you can use the functionality available. We will now talk about using the different functionality available split up by the functionality:

- Identity Functionality
    - This functionality is only available via the Administrator page when signed in as an Admin user. You can get to the page on the left side navigation menu. It will allow you to add or remove accounts from the database.
- User Notification Functionality
    - This functionality will show all active notifications to the users so they can notify staff about outages in the Rescue21 system. It is available on all pages by clicking the yellow notifications button in the top right corner of the screen.
- Admin Notification Functionality
    - The admin notification functionality will also list all of the active notifications except they can flip the activation status which will stop displaying the notifications on the website. This is meant for small status flips but if you are flipping more than 10 notifications then I'd recommend flipping the activation status in the database instead.
- List View
    - This view is available via the "List View" button on the left hand side navigation menu. This view will generate weather summary views of the sites available. You can flip through the pages to see the different area of responsibilities.
- Map View
    - This view is available via the "Map View" button on the left hand side navigation menu. This view will show the status and geological location of the SWAPR devices across the United States.
- Historical View
    - This view is available via the "Historical View" button on the left hand side navigation menu. This view will create one of three graphs that contain averaged data from start date to end date over the provided interval. You can create a wide variety of graphs by changing the site to use, the graph to use, the start and end date, and the interval. Recommendation: stick to smaller sets of data which are easier to see. If you need a large period of time then use a large interval to average over. To get the best visualization, select only one or two data attributes at a time.
- CSV Data Exporting
    - This functionality is only available after a graph has been created using the historical view. Once the graph is created, you can download the data used to generate that graph by clicking the "Download CSV" button at the bottom of the page.

Maintenance

Our system will generate a lot of data (1 entry every 5 seconds w/ Notifications created whenever the status of anything but green is found). For this reason, it will be necessary to "clean" out the database where you empty all the data out of the database that the website is connected to and delete it or put it in another database that isn't accessed as much. We only have one database, so for our implementation of the project, we would just delete the data in the database using this mysql command:

"DELETE FROM ebdb.Entry WHERE Id > 0;"

You could also create a snapshot on AWS which can easily be done in the user interface of AWS. You can learn more about doing this at https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateSnapshot.html. There is also the option to create a backup of your database in AWS. More can be found about backing up your database here: https://aws.amazon.com/getting-started/hands-on/amazon-rds-backup-restore-using-aws-backup/.

The other maintenance factor to consider is updates to some of the few dependencies that we use. The main dependencies that I am referencing are the wrapper classes of Chart.js and Canvas.js (ChartJs.Blazor.Fork and Blazor.Extensions.Canvas). These packages are made specifically for Blazor Server independently from Chart.js and Canvas.js. If you run into a problem with these dependencies or any other dependencies in the project then you will need to update the nuget packages. You can find instructions on doing this at: https://docs.microsoft.com/en-us/nuget/consume-packages/reinstalling-and-updating-packages. Updating dependencies may break the code which can only be fixed by manually modifying the functionality that broke to meet the new requirements set in the updated version of that dependency.

Last but not least, there will be updates to the AWS environments required whenever AWS decides to update their products. Specifically, if they update their SQS, SNS, Lambda Function, RDS, EC2, Elastic Beanstalk, IAM, or VPC products then there could be issues that arise from AWS. Additionally, any modifications to the code could cause unexpected issues in the hosted environment. For example, after making changes to the project code and publishing on elastic beanstalk, somehow a .dll file was deleted which completely broke the historical view. In order to fix this, we had to ssh onto the ec2 instance hosting the elastic beanstalk environment and install the .dll package again.

Troubleshooting

This section of the page will focus on problems that you may have encountered during the duration of this user manual. This part will be separated by the subsystems: Simulator, Reader, Database, Website, and Orchestra.

- Simulator
    - Timeout
        - There are a number of reasons as to why the Simulator would timeout. Mainly, it is because the Reader has stopped working. This could be because it has erred and needs to be restarted. It also could be because of the issue named “Simulator/Reader Timeout” in the Known Issues and Recommendations section.
- Reader
    - Timeout
        - There are a number of reasons as to why the Reader would timeout. Mainly, it is because the Simulator has stopped working. This could be because it has erred and needs to be restarted. It also could be because of the issue named “Simulator/Reader Timeout” in the Known Issues and Recommendations section.
- Database
    - No Data
        - This could be happening for a number of reasons like the database is not responding, it’s pointing to the wrong datatables, you are not being correctly authenticated, or there is no data in the database to get. The best way to troubleshoot this issue is to check that the database is up and running, that the connection string works through a program that will error if it’s incorrect (e.g. MySQL WorkBench), and viewing to make sure that there is data in the database.
    - Timeout
        - This could be happening because the database server is offline so the website is unable to communicate with it. You should check that the database is online and functioning. Please look at the no data instructions for other checks you can run.
    - Foreign Key reference unknown
        - When adding entries or notifications to the database, you need to have foreign key references. That means that every entry added to the database will have to have a site id to reference first. It also means that every notification added to the database will have to have a site and entry id to reference. If you don’t do this then this error will be displayed. For example, if you run the orchestra for sites 1-100 then it will error because there are only 37 sites in the database.
- Website
    - Disposed Object / Double Operation / Canceled Operation

        - This means that some object that was being used has been called on again after being disposed of, was already used, or is in the middle of an operation and didn't finish. This error should be fixed, however, it might show up depending on how rapidly you are making calls to different functionalities on the website. For example, if you clicked on the list view and waited a half second then immediately switched to the map view then this might cause a disposed object error which will break the site because it was in the middle of a database call before being interrupted
- Orchestra
    - Com Port already in use
        - If this message or similar message is shown when setting up the com ports for the Orchestra then the site that would have been simulated on that pair of ports will not work. Please refer to the Known Issues & Recommendations section under the name "Orchestra Com Port Overlap" for more information.

Known Issues & Recommendations

This section will describe the known issues and recommendations on solving those issues. The point of this section is to help direct the next team of developers in modifying this project into a production quality product.

Issue Name: Admin Notification View Out-of-date
Description: Admin notification view updates user notification view but if there are new notifications then the admin notification view will be out of date compared to the user notification view.
Cause(s): Flipping the activation status of a notification on the admin notification view.
Effect(s): User notification view is more up to date than admin notification view.
System Affected: Notification Views
Information Known: This can be fixed by having the admin notification view call getNotifications when calling the event that updates the user view.
Error Message(s):
Known Fix(es):
Notes:

Issue Name: Historical View Dataset Intervals Not Consistent
Description: The user provides a start, end, and interval period when calling historical view which should be used to break up the data to average over whatever interval period. The problem is that the way we choose to solve the problem of finding the interval data belongs to doesn't find the correct starting interval period. This can lead to inconsistent ranges of time where data is starting from and ending on which can throw off the data calculations. The way to fix this problem is to come up with a better method for getting the interval that an entry belongs to. This function is found in the DataManager in the GetEntryInterval function on line 1079.
Cause(s): Calling historical view (Which will call averaging functions which call getEntryInterval)
Effect(s): Datasets have interval start times that are not following the interval set.
System Affected: Historical View
Information Known: getEntryInterval is causing the issue by finding DateTimes that do not fall in the set of interval times.
Error Message(s): No error messages given.
Known Fix(es): The GetEntryInterval function is splitting in half but it isn't following the nearest interval. The easiest way to fix this issue is to find the workingMiddleStartTime by dividing the range between start and end time by 2 and getting the closest starting interval time near that point. This would make sure that the DateTime returned is always in the set of intervals.
Notes:

Issue Name: Map View Clicking On Historical View
Description: This Issue occurs when clicking on a marker from the map view. What happens is the new page that opens contains the historical view but at some point the original InvokeAsync call that created the historical view page will abort which will cause the website to error.
Cause(s): Closing the new window. / Opening historical view & downloading csv / Randomly failing when starting new operations
Effect(s): Error b/c task was canceled
System Affected: Map View & Historical Views
Information Known: the NavigateInNewTab function (line 61 in MapView.razor) is responsible for causing the website to crash instead of logging the task aborted event.
Error Message(s): Task was aborted.
Known Fix(es): This could be fixed by reverting the functionality so that the map view will take you to the historical view without keeping the map view open. You can also add a try catch in the NavigateInNewTab function which will prevent the error from crashing the website.
Notes:

Issue Name: Identity Registering New Users
Description: We suspect that the issue is something to do with the fact that we converted the datatables from SQL to MySQL without changing any code except for the Identity database connection string. This could be a good thing because no one can access the dev site unless they know the credentials. However, this will be bypassed completely when integration into GDMS' Active Directory service is done. When you try to register new users, it will create an account. However, it will not be added to the user group so the new user will have no permissions on the website.
Cause(s): When trying to register new users
Effect(s): User is added to the database without a user role.
System Affected: Identity Functionality
Information Known:
Error Message(s): No error message given
Known Fix(es): Only known fix is to force add the account to the user group. This is how both the Administrator and User accounts provided were created. You can change the accounts forced in the Appsettings.
Notes:

Issue Name: Orchestra Com Port Overlap
Description: When using the Orchestra, you can overlap with Com ports that are being used by other devices on your computer. When this happens, that connection will not work so that site will not generate data.
Cause(s): Running Orchestra
Effect(s): Com ports will overlap causing the site given those ports to fail
System Affected: Orchestra

Information Known: This is caused because of the incremental approach that we took to creating the com ports. We don't check if one exists or try recreating ports, we just create ports from the starting com number to the starting com number + (number of pairs created * 2).
Error Message(s): Com Port in use / Timeout errors
Known Fix(es): The only fix to this problem is to modify the Orchestra so that it checks if the port is being used and then creates a port or moves on to create another port. It will then need to remember the ports not used and skip them when assigning ports to the simulator and reader pairs.
Notes:

Issue Name: Simulator/Reader Timeout
Description: The Simulator and Reader subsystems will timeout when they are started with a large interval between the start of the Simulator and the start of the Reader. This happens even if it is working for some time because it will slowly shift until it will timeout.
Cause(s): launching the Simulator and Reader with an interval of time between starting them
Effect(s): timeout errors will occur usually.
System Affected: Simulator and Reader
Information Known: Only really happens when running them through visual studios and not starting them really soon after the other
Error Message(s): Timeout exception
Known Fix(es): The only fix that I have seen is by having both projects loaded and on two screens. You then start the reader before the simulator. However, the other fix is to use the Orchestra when launching the system.
Notes: This issue doesn't seem to be a problem when using the Orchestra to launch the two subsystems.

Brief File Description

This section will briefly describe the files that are in the website to help reduce the time required to find where errors may occur during development.

- Wwwroot
  - ChartJsBlazorInterop.cs - dependency for Chart.js
  - csv/entriesExport.csv - The output file for the csv file (can be changed in config)
  - MapData/googleMapTestV3.svg - The output file for the result map (can be changed in config)
  - MapData/USA_Cropped.svg - The input file for the map view (can be changed in config)
- Areas
  - Identity - Holds the pages that are used for the Identity functionality.
- Components
  - NotificationModal.razor - Used for the flipping of status on Admin notification view
  - NotificationStatusModal.razor - Used for the user notification view
- Data
  - ApplicationDbContext.cs - Identity Db Context
  - DataManager.cs - Does all of the data operations occurring on the site
  - Entry.cs - The database model for Entries
  - EntryDbContext.cs - Site / Entry / Notification Db Context
  - HistoricalViewOptions.cs - Used to contain the form data from the historical view homepage
  - ListItem.cs - Used to separate sites in the notification view
  - Notification.cs - The database model for Notifications
  - NotificationComparer.cs - Used to order the notifications for notification views
  - NotificationManager.cs - Manages the notifications
  - Site.cs - The database model for sites
- Data/Configs
  - CSVConfig.cs - Configuration data model for CSV Data Exporting
  - IdentityConfig.cs - Configuration data model for the Identity Functionality
  - ListViewConfig.cs - Configuration data model for the List View
  - MapViewConfig.cs - Configuration data model for the Map View
- Pages
- /* Identity Page */
  - Administrator.razor - The page that can be used to see all the accounts on the website
- /* Admin Notification View */
  - AdminViewNotification.razor - The Notification View for Admin Users
- /* List View Page */
  - ListView.razor - The List View Summary Page
- /* Map View Page */
  - MapView.razor - The Map View Summary Page

- /* Historical View Pages */
  - HistoricalView.razor - The Historical View Home Page
  - LineHistoricalView.razor - The Line Historical View
  - BarHistoricalView.razor - The Bar Historical View
  - RadarHistoricalView.razor - The Radar Historical View
- /* CSV Data Extraction Pages */
  - Download.cshtml - The Download Page that redirects to Download.cshtml.cshtml for the CSV Data
  - Download.cshtml.cshtml - The logic behind the download page for the CSV Data Extraction
- /* System Generated Pages */
  - _Host.cshtml - Holds page imports for the website
  - Index.razor - Home page of the website
  - Error.cshtml - Home page of an error
  - Error.cshtml.cs - Home page of an error logic
- /* Development Page */
  - FetchData.razor (Not Accessible From Nav Menu) - Can be used to check data in database
- Shared
  - LoginDisplay.razor - Identity Functionality
  - MainLayout.razor - Holds the Identity Page Locking and User Notification View
  - NavMenu.razor - The navigation menu shown on the left hand side
- _Imports.razor - The imports given to every page
- appsettings.Development.json - Development version of configuration data
- appsettings.Production.json - Production version of configuration data
- Appsettings.json - The default configuration data that redirects to the version of the project
- Aws-beanstalk-tools-defaults.json - Aws configuration data
- IListExtensions.cs - Dependency for Chart.js
- Program.cs - Builds the program
- SampleUtils.cs - Dependency for Chart.js
- Startup.cs - Defines the start up configuration

Conclusion

We would like to thank once again for choosing the Rescue21 SWAPR Data Dashboard for your organization's needs. We hope that you enjoy many years of service from the Rescue21 SWAPR Data Dashboard. Team S.A.R.C.I. has had a joy producing this project and working with you over the last year. We want to thank you for being an awesome client!

While we are all moving on to professional careers, we would be happy to answer short questions in the coming months to help you get the product deployed and operating optimally in your organization. Feel free to reach out to us @ dsw235@nau.edu.