CS Capstone Design

Technical Demo Grading Sheet (100 pts)

TEAM: _____Search And Rescue Coastal Intelligence_____

Overview: The main purpose of the "Technical Demos" is to very clearly communicate the extent to which the team has identified key challenges in the project, and has proven solutions to those challenges. Grading is based on how complete/accurate the list of challenges is, , and how convincingly and completely the given demos cover the given challenges.

This template is fleshed out by the team, approved by CS mentor, and brought to demo as a grading sheet.

Risky technical challenges

Based on our requirements acquisition work and current understanding of the problem and envisioned solution, the following are the key technical challenges that we will need to overcome in implementing our solution:

C1: Storing generated data in the database.

Overall, accessing stored data from the database and displaying it in a beneficial way that also meets our clients requirements will have its challenges. The steps to generating data and storing the data into a database is the first challenge we need to overcome to allow us to continue to following challenges. Generation of the data will require a script which writes text files with appropriate parameters. We may also want to test a case where the data in the text file, the simulated Site Weather and Power Recorder (SWAPR) device data, is problematic. Once the data is generated, it needs to be sent to the database using a TCP connection.

C2: Processing data from the database and generating a graph in R.

To retrieve the data from the database and use it to generate a graph with the R programming language will be our challenge 2. The database will need to be queried to access stored data, then the data will need to be passed into the class that handles creating graphs. This class will have R integration using a built-in .NET feature. Once the graph is generated it will need to be displayed in the user's environment.

C3: Creating a display with JavaScript using data from the database.

The final challenge we face is creating a list view of a SWAPR device widget using JavaScript with D3.js. Similarly to creating a graph, data must be retrieved from the database. In this challenge the data will be passed to an integrated JavaScript script in our C# program. Usage of JavaScript in C# in .NET uses JavaScript Inter-polarity injection functionality to use JavaScript scripts in C#. The widget should show accurate values relative to the pulled data from the database and be clearly visible in the user's environment.

Challenges covered by demos:

In this section, we outline the demonstrations we have prepared, and exactly which of the challenge(s) each one of them proves a solution to.

Demonstration 1: Populating the database with simulation data

Challenges addressed: C1

Flight Plan:

1. Generate data that emulates a SWAPR device
2. Open TCP connection to a database
3. Send data to database using TCP connection
4. Store data in database

Evaluation:

✔ Generate SWAPR data to test our system with realistic datasets.
✔ Open and send data over a TCP connection to test communication with a database using a real protocol.
✔ Store data in a database to give our other challenges usable data.

✔ Other evaluative comments:

Demonstration 2: Create a graph

Challenges addressed: C2

Flight Plan:

1. Access the database and store data in temporary object
2. Use the data from the object in the R section of the code to generate the graph
3. Display the created graphic

Evaluation:

✔ Accessing the database and acquiring data tests an important action in our system because it is how we use the data.
✔ Generating a graph to test the core action in producing visuals for our website.
✔ Displaying the graphic tests the ability to present the image to a user.

✔ Other evaluative comments:

Demonstration 3: Create a list-view

Challenges addressed: C3

Flight Plan:

1. Create a basic graphic for the list-view widget – like our examples. A rectangle with the name of the device, wind direction, wind speed, temperature, and humidity default values displayed for a template.
2. Access the database and retrieve data then store it in a temporary object.

3. Insert the values into the list-view widget so the various values on the graphic are updated.

Evaluation:

✔ Created the list-view widget graphic to test displaying the widget.
✔ Access to and retrieval from the database is a key action needed for this system.
✔ Inserting update values into the widget needs to update the values displayed so the user can get updates through the interface.

✔ Other evaluative comments:

Other challenges recognized by not addressed by demo:

If there were challenges you listed earlier that were *not* covered by a demo, list here. This will hopefully be a short list…but better to be clear about where you are. If you have items here, you could list (if applicable) any pending plans to reduce these risks.

- **Sending and reading data over virtualized com port on RS-232 protocol**
    - Our team is not concerned with testing this for the flight plan, because the virtualized com port communication can be done using the Modem com0com tool. This tool can be downloaded and used to create any number of virtual COM ports as viewable signal lines between our Simulator and Reader programs once we have those created.
- **Connects any number of simulator and Reader Subsystems over virtualized com ports for lab environment stress testing**
    - This is a stretch goal for our team. We plan to do this using an orchestrator, but will not put time into it until we finish our clients main concerns.
- **Establish a TCP Host connection with the Reader Subsystems**
    - The Reader Subsystem will be relatively simple to achieve with past experience of the team. For that reason we are not focused on creating the reader or establishing any TCP connection until we have proved the harder systems which we will cover in our demo.
- **Hosting the database server off Amazon Web Services Relational Database Service (AWS RDS)**
    - AWS RDS is an expensive service to run which is why we have done the testing to ensure it will work but we have not decided to use it for our demo because we

would have to pay for it out of our pockets. We will instead establish a local database and querying it for testing. Once we have finished our flight plan and the interaction with our database and Blazor is connected and understood, we will host the database on the AWS RDS with the budget given to us by our client.

- **Hosting Blazor Server on Amazon Web Services (AWS)**
    - Along with hosting our database on AWS, we will host the Blazor Server. This and our database depend on each other and have been tested but we will not demonstrate hosting in this demo because we would have to pay for it out of pocket.
- **Queue TCP data collected in Amazon Web Services Simple Queue Service (AWS SQS) and create database entries from data**
    - The AWS SQS has been tested but is another expense that we do not want to pay for ourselves which is why it has not been implemented in this demo. Once our client funding is applied to an account, we will start using AWS SQS.
- **Secure authentication between subsystems as well as "faking" user accounts and role based permissions; Setup Windows Active Directory to manage user accounts and role-based permissions;**
    - These security and management related functionalities are not what our client wants us to focus on first. This will be a challenge to solve once we complete the stated flight plan and have established the website with Blazor on AWS. Also, Dylan has past experience with "faking" user accounts and role based permissions on .NET so this will be relatively simple to implement after we have the main functionality.
- **Ability to query the database server in a secure manner**
    - Once we have set up all of our usage with AWS this will inherently be the case for our system. The connection between our database server and our queuing service will be guaranteed secure by communicating through AWS VPC which is a private subnetwork with strict security protocols.
- **Notification system that detects when there is a problem with an RFF and creates a modal for a specific user group**

    - This is a straightforward process that we can test once our database and Blazor server is hosted on AWS. This will effectively be a if then statement that will create generic messages which is why we are not concerned with proving it in our demo.
- **When in the graphical view of historical SWAPR data the user should be able to export the selected data to a comma separated file**
    - Exporting data to a text file will require data from the database that we can then write to a .csv and store locally on the user's system. This is something that multiple team members have already done so it isn't our biggest concern for our demo.

Our plan, once the flight plan we laid out is complete, is to test these next challenges. The functionality that our client wants most is what we are trying to prove in this demo. Our main goal for this demo is to prove the functionality of the packages that we choose within the Blazor Server architecture which may prove difficult. Other challenges we know we will face involve setting up security rules for AWS and establishing a connection from outside AWS that keeps our system secure.

Our priority at the moment is to deal with the challenges of creating graphs using data and visuals for the website. In doing this we are also testing the compatibility between our programming languages and tools we plan to use: Visual Studio, Blazor, R, and JavaScript.