**ScavengeRUs**

**Code Documentation**

SCRUManeers (Team 2)

By: Brandon Bowers, Dalton Weems, William Simmons

Tyler Ebb, Dyson Simonds, and Ryan O'Connor

**Glossary**

1. General Information
2. PID
3. Product Report
4. Process Report
5. Prototype Diagrams
   1. Use Case Diagrams
   2. Flow Graphs
6. Page descriptions and functions.
7. Instructions and Recommendations for the Future

**General Information**

ScavengeRUs is a scavenger hunt based around using QR codes and a device with a camera to scan items and track them with that device. It must not use cookies to save any information on a device, so everything must be stored using a database with a web server hosting the program.

Everything in this Word document is meant to document everything that our team used/made throughout the project. What we made doesn’t have to be used, and there are a few minor differences in the code compared to the prototypes due to the different code changes as it was developed.

**Software**

Our team used a variety of software and resources to complete a phase 1 prototype of the project. These include:

* **Blazor** – Open-source framework for web server development. This is the framework for our code.
* **Visual Studio 22** – This version is required due to .NET 6.0 being only available on this at the current time.
* **GitHub** - https://github.com/Simmonswa/ScavengeRU
* **Trello** – This site allows your team to manage objectives like a bulletin board. We used this to keep track of who accomplished what tasks and what still needed to be completed. We also stored links to shareable documents within the Trello Board.
* **Clockify** – This is a mostly free website that allows for an easy way to track time within a team. It can create simple charts with the time added in by team members.
* **Diagrams.net** - Free open-source website that allows anyone to create charts and diagrams. Highly recommended to use this instead of Lucid charts.
* **Azure Webserver –** The website was implemented using an Azure Webserver which can be implemented using visual studios.
* **SendGrid –** Email API service
* **Vonage –** SMS API service

**PID**

|  |  |  |
| --- | --- | --- |
| **Project:**  **Team:** | ScavengeRUs  SCRUManeers | **Business Objectives:**  This project will enable third parties such as businesses, charities, or communities to engage with local individuals and groups for scavenger hunt style events using QR codes.  **Mutual Benefit:**   * We receive monetary gain from creating the program along with hosting and customizing scavenger hunts to fit the client’s needs. * The local businesses are expected to receive an increase in foot-traffic, advertising, and sales due to exposure. |
| **Client:**   * Mr. Kinser * [kinsersretired@gmail.com](mailto:kinsersretired@gmail.com)   **Sponsor:**   * ETSU * William Rochelle * [rochellew@mail.etsu.edu](mailto:rochellew@mail.etsu.edu) * Nicks Hall Room 487 | |
| **Vision Statement:**  We want to give businesses and organizations an innovative, subscription-based method for increasing their exposure and revenue utilizing a scavenger hunt style application played by locals. | | |
| **Scope:** | | |
| **User Roles:**   |  |  | | --- | --- | | * Player/Team:   + Requests unique invite link/code.   + Scan QR code.   + View scavenger hunt progress.   + View QR code location and location information. | * Host: * Requests scavenger hunt to be hosted. * Create QR codes and/or assign QR codes to scavenger hunt. * Create scavenger hunt settings and permissions. | | | |
| **User Functionalities:**   * All users shall be able to click a login button or register button in order to gain access. * Players shall be able to click a button and enter a unique code into a text box in order to join a scavenger hunt session. * Players shall be able to click a button to enable the camera to scan QR codes. * Players can view the current progress of a scavenger hunt session. * Players can select a scavenger hunt location and view information about the location. * Players shall be able to select their player name and change it. * Hosts shall be able to view the list of scavenger hunt items. * Hosts shall be able to edit individual scavenger hunt item information. * Hosts shall be able to view and share the game session access code. * Hosts shall be able to edit additional settings and permissions of a scavenger hunt session. | | |
| **Special Issues/Considerations:**   * The QR codes must be placed in wheelchair accessible areas and within reach of these players. * QR codes should be monitored to ensure they are not missing or damaged, and a feature will be included to report QR location problems. * Areas with a QR code should be easily visible and in safe locations without nearby hazards. | | |
| **Glossary of Terms:**   1. **Scavenger Hunt** **–** Refers to a ScavengeRUs instance. 2. **Access Codes** – Refers to a multiple character code to invite users to join the same scavenger hunt. 3. **QR Codes** – Refers to codes used to scan and collect for a scavenger hunt. 4. **Scavenger List** – Refers to list of QR codes or scavenger hunt items used for a scavenger hunt instance. 5. **UI** – Refers to user interface for interacting with multiple functionalities in an application. 6. **Database** – Refers to a collection of data that is used concurrently with the application. | | |
| **Priority List of User Functionalities:**   1. Create a rough UI outline of the application to test functionality. 2. Implement SQL database to hold QR data. 3. Implement actual QR code scanning. 4. Implement Progression within scavenger hunt. 5. Implement User(s) SQL Database. 6. Implement Access codes/ Congrats message. 7. Implementing name editing. 8. Polish UI to be more user friendly and appealing | | |

**Product Report**

Finishing the final iteration of phase 1 of the SCRUManeer’s ScavengeRUs project, we have completed several goals for the project. We have a current webhost for the project and have a working database along with it.

The application is running off .Net 6.0 using Blazor as a webserver template and uses HTML, C#, CSS, and ASP.NET. The database uses MySQL. The home page prompts users to enter display name, email address, and phone number. Once the user submits valid information, they are created in the database alongside the tasks they need to complete. A text message is sent to their phone with the access code, and an email is sent with a link to the game page. The Players, Tasks, and PlayerTasks pages are currently used as a test environment for the database. The game page is used to display tasks and task information. QR scanner is currently functional but will only display the information that is stored on the QR code without linking to the database.

There were some functionalities and desirable changes that did not come to fruition this semester. We did not get users authenticated by their access code. We did not connect our game page to PlayerTasks table as desired. The QR code reader is there, but we do not have the functionality for it to update PlayerTasks. There were some general CSS changes we had hoped for as well such as resizing div containers and filling out the whitespace on the home page. A new logo or font for the application name could improve looks as well.

Some recommendations that are not 100% necessary would include using Azure to send SMS and Email. Having 1 service instead of 2 for these functions would be more beneficial, and it’s the same provider as the server hosting. We ran out of a free trial for SMS and Email, so we switched to SendGrid and Vonage. The biggest future recommendation I can give would involve changing how the database is implemented. We did it manually in MySQL Workbench, but it made it very difficult for anyone else to work on things. Try using ASP.NET built in ORM to map out models to a database so any future changes anyone could do in the progress and the database would update.

Step-by-step instructions will be at the bottom of this deliverable.

**Process Report**

**What Worked**

Our team worked well doing collaborative coding in group meetings. We often worked in pairs of 2-3 while coding to help with errors or questions. Trello boards helped our team remember what else needed to be completed as well as keep track of who has completed what, this allowed us to know who to ask for certain snippets of code if the need were to arise. Another aspect that helped us was discord, there was always someone available to help with any questions during development and it worked great for meetings and voice calls. Our management on the project was decent, we used GitHub as our code repository, and it served us well with updating our code nearly effortlessly.

**What Did Not Work**

Our team had issues in properly breaking down the code into sizable chunks for individuals. This issue mainly occurred due to not having a singular end goal for what the program would look like and achieve. As the program advanced, this issue diminished as team members could work on specific webpages without interfering with others work when pushing the code into one in GitHub. Another issue was GitHub itself, we had issues with having to recreate our repository several times and a major issue with locking our account when our repository was set to public and not private. One thing I would like to have our group done was have a mandatory daily meeting to discuss where we stood. Creating the database earlier on in the iterations would be more ideal than at the end. We could not press forward with functionalities until a lot of time was spent fixing up the database.

**How our team worked**

Our team mainly used Trello for progress tracking and updated what needed to be worked on. Each person usually picked what task they wanted to work on and updated as they went along, there usually were not any fixed requirements on who needed to work on what it was mostly a group effort when needed. Our Clockify did a good job showing who worked on what and when, and if one person felt they needed some relief they asked for help when needed.

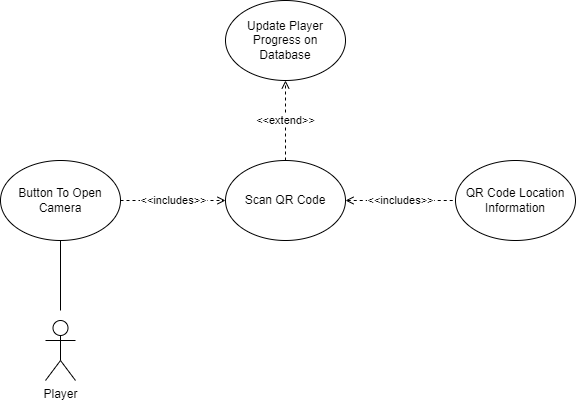
**Prototypes**

**Important!** These are all early prototypes/ideas for how the code should function. Most of these did not get implemented or got massive changes.

**Use Case Diagrams**

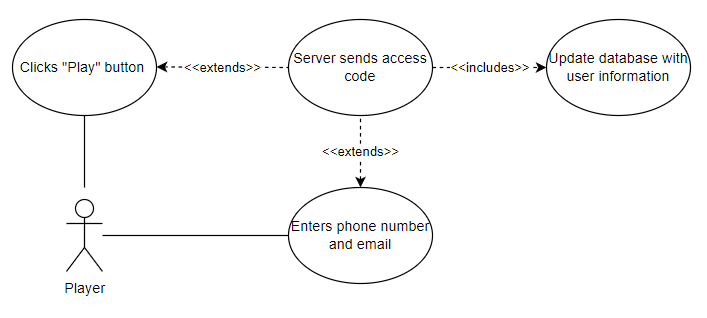
**Use Case 1:**

Player scans QR code



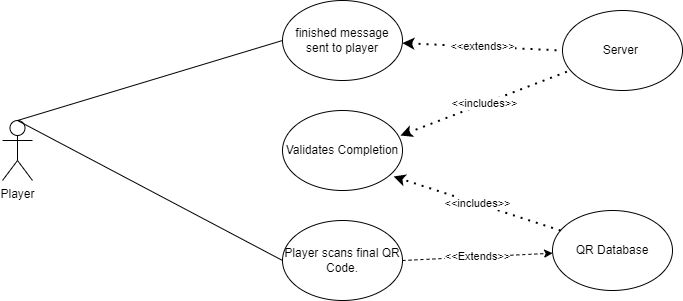
**Use Case 2:**

User requests an access code.



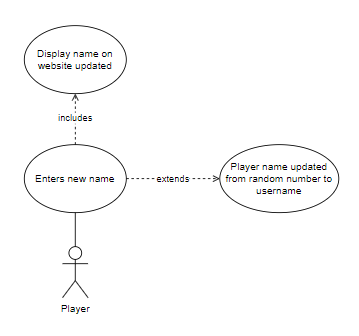
**Use Case 3:**

User finishes scavenger hunt.



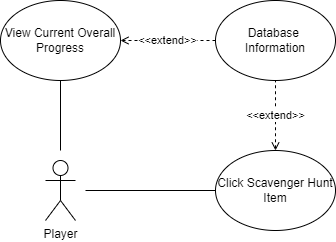
**Use Case 4:**

User changes display name.



**Use Case 5:**

Player checks scavenger hunt progress and information



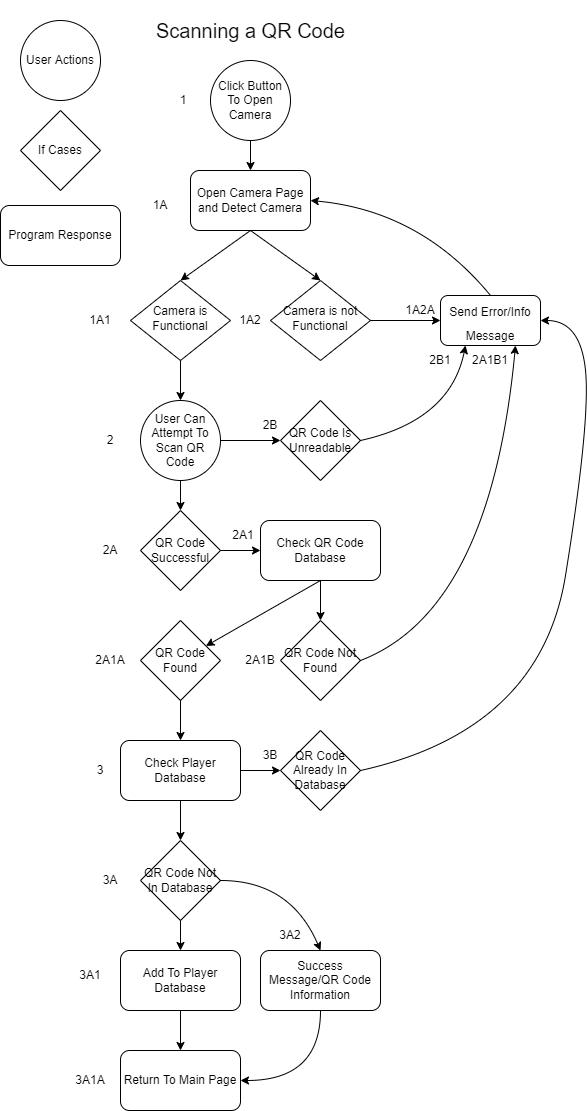
**Flow Graphs**

These graphs are meant to numerically graph the flow of individual actions that need to be performed by the program or user, with if cases separating different possibilities. (.drawio file can be used to edit it in diagrams.net)

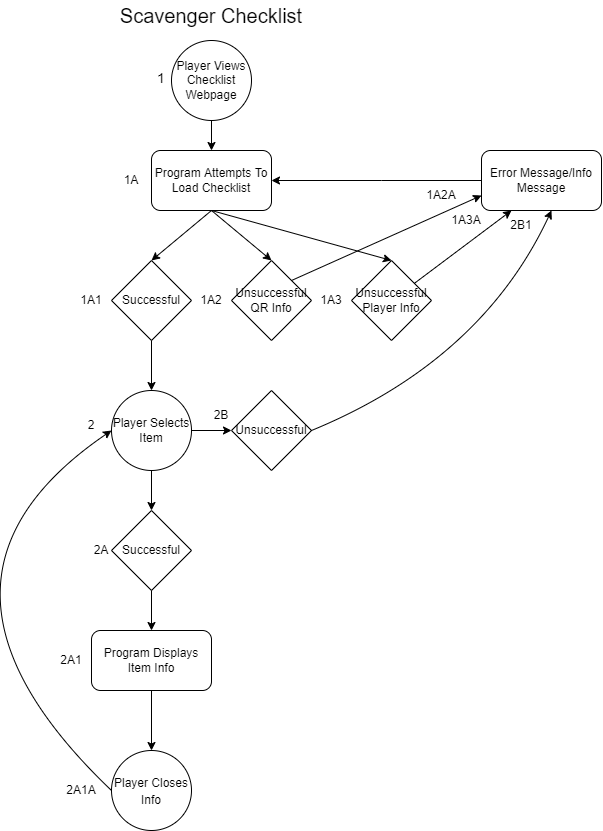
* Success Cases Follow Left Side
* Errors, Negative Results, and Additional Checks
* Unique Numbers Given To Each Response, Either Program Or User Response
* Number System Returns To Single Digit During User Actions Or Unique Program Actions

Minimized the flow graphs to save document space. Original png files should be saved in folder or png size can be increased in Word.

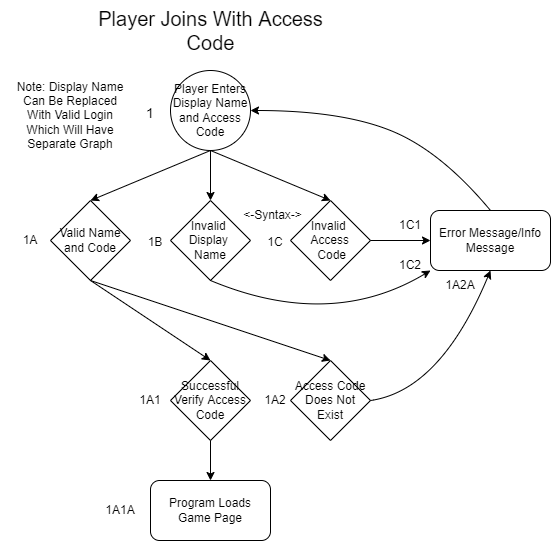
Flow Graph 1



Flow Graph 2



Flow Graph 3

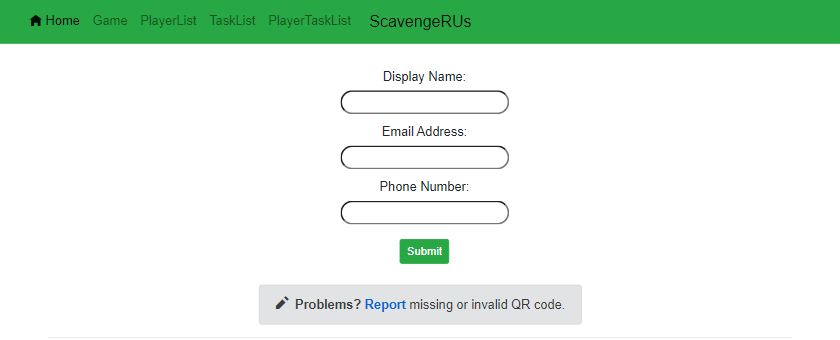


**Code**

**Web Pages**

The webserver is hosted using Azure Webserver which can be accessed and run using Visual Studios.

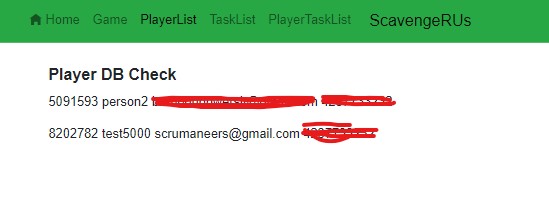
**Home page – Index.razor**

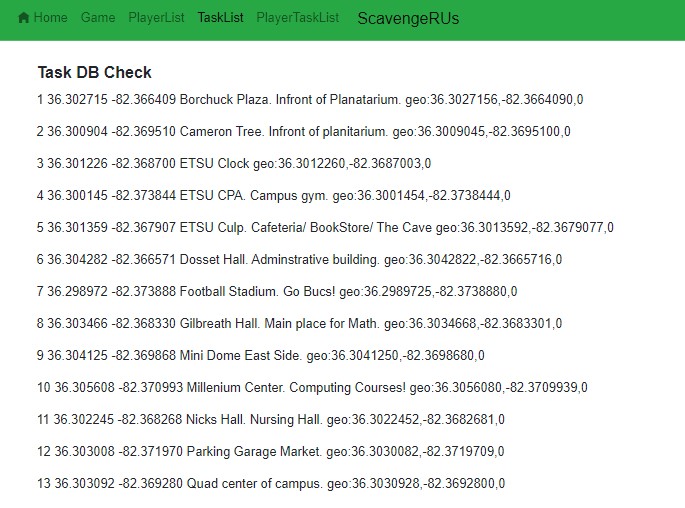


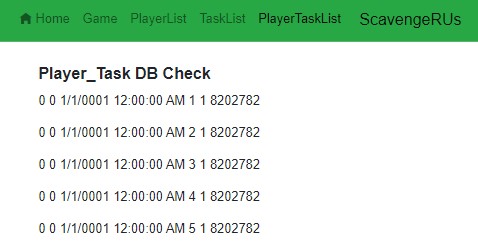
The home page, also considered Index.razor, is the first page that users are supposed to encounter when loading ScavengeRUs. Clicking “Submit” creates the player in the database and populates PlayerTasks with their required tasks.

**DB Test Pages – Players.razor Tasks.razor PlayerTasks.Razor**

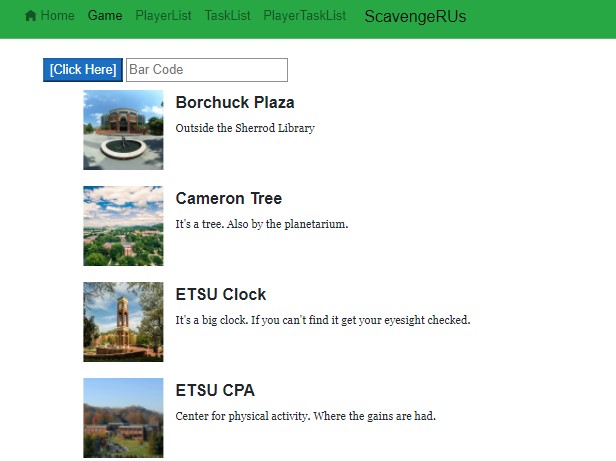
These are used to test/display current information in the database. It currently states loading as that is the state designated to it when the database is empty and has nothing to display. In cases where there is information to output, it will display Players, Tasks, and PlayerTasks info. Player information displayed is access code, display name, email, and phone number. Task information displayed is task id, latitude, longitude, task info, and qr code. PlayerTasks information is Player\_TaskID, is\_complete, time, task\_id, game\_id, and player\_id. Examples of these 3 test pages are displayed next:





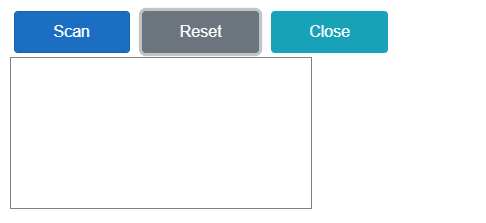


**Game Page + QR Scanner – Game.razor**



The game page shows the lists of tasks designated for the game session the player is in. It displays the list and shows a progress bar on the bottom.

The QR scanner will scan a QR code through a camera on the user’s device. Pressing [Click Here] opens the primary camera set by the device.



The white box is where the output of the camera will be displayed. (It is currently disabled for privacy)

**Additional Page Information**

There is a file designated for reporting missing QR codes in case they might go missing on the location. The file is designated as Report.razor and is currently blank.

PlayerDetails.cs is used to get the player Display\_Name and share it throughout all the webpages. This, however, does not work, as saving players still isn’t fully implemented.

**Data**

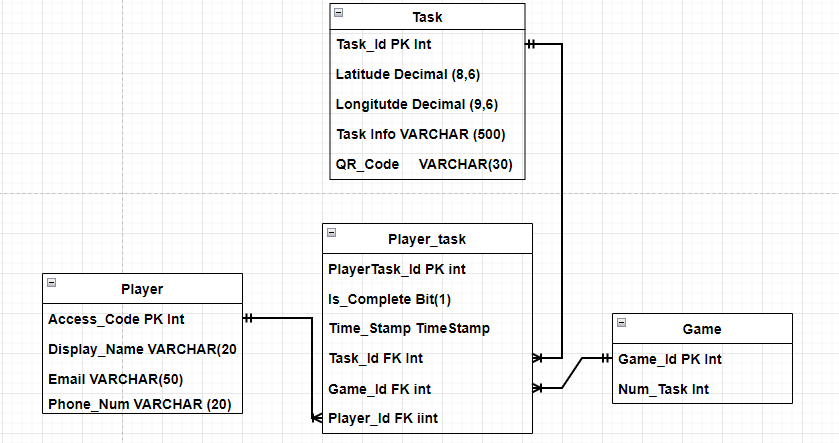
**RegisterModel.cs**

This is used with the Homepage for user parameters. It sets the requirements for the input boxes as well as sets and gets the information for all the input boxes.

**SmsService.cs**

This is used to send a text message to a phone entered by the user on the home page.

**Model**



This model is the Entity Relationship Diagram for the database.

**Instructions and Recommendations for the Future**

* + - 1. **Deploying to Azure Server**

The hosting software used for our needs was Azure. AWS could work too, but it's painful, and GitHub pages/firebase are out because they require a static format. Signing up for a free student account should let you host for 12 months free, which is more than enough for the semester. I recommend the following order for deploying as painlessly as possible:

1) Set up an Azure account and create a server first. The control panel will be intimidating: 85% of it is useless for this project.

2) Pull your repo from GitHub and open it in Visual Studio ‘22.

3) Click publish, Azure, sign in, and it should compile and publish quite handily.

4) If you don’t want to do that again, there’s steps you can take with GitHub actions to auto-deploy from committing to your repo's main branch. Google will help you there. With luck, you’ll never have to touch the server control panel again if this works. If not, you can manually upload from Visual Studio whenever you need to deploy.

5) A database is required for further iterations in the spec. Luckily, azure can also host one of those. The In-App SQL will not fit the requirements, so string up a second server.

6) Use MySQL to populate the server with tables you need. We should have attached the ERD to our solution somewhere above.

7) On the SQL server control panel’s home page, there should be information for a connection string available. Copy this, and import it into your appsettings.json. Line 11, titled “default”.

* + - 1. **Email and SMS Service**

FIRST AND FOREMOST, DO NOT UPLOAD API KEYS OR ACCOUNT INFO TO A PUBLIC GITHUB. USE GITHUB SECRET KEYS OR KEEP THE REPO PRIVATE. The email service used for our needs was SendGrid. The SMS service used was Vonage. They both offer a free service that can fit the scope of this project. I would highly recommend looking at Azure’s email and SMS service so it’s all under one provider. If you do choose to keep Vonage and SendGrid here are the steps needed to make it work:

1) Set up an account on both websites Vonage and SendGrid

2) Vonage:

a) Create an application on their website for organization.

b) Apply the provided sender phone to the application

c) Place API key, Secret Key, and Sender Phone in SmsService.cs

3) SendGrid

a) You will need to gather the API Key and a Sender email that you want the game emails to be sent from.

b) Place this information in Index.razor in the SendEmail method.

c) Add game link to email content

4) The email and SMS service should be operational if the Index.razor form is still the same. It takes the information from this form and uses it for both services.

* + - 1. **Recommendations**

Here is a list of recommendations for the application and working as a team in general:

1. Discord is a must for communications and online meetings.
2. Trello is great for organization and keeping track of who does what.
3. Clockify is the best free time-tracking software. You can export many graphs to explain what people were working on.
4. Start early in iterations. Problems will appear every time and you will wish you had more time to get questions answered.
5. Meet often even for small amounts to explain what has been done and touch base on what everyone will be working on.
6. Understand that some people can look at a Trello board and get to work on something, where other people may require the group or leader to assign them work. One way does not work for everyone in a group, catch these problems early!
7. Blazor is great. None of us had experience with it, but it is easy for everyone to learn so don’t be intimidated by new things!
8. I highly recommend using DBcontext, migrations, and ORM to make the database so anyone in the group can change the context to update the database. We had 1 person do it manually outside the project and it required all of us to go through him to find the design and names of entities. If there is a problem and they are not around the application comes to a standstill. So highly recommend switching to the method above. Do database earlier on in the project, do not save it for later!
9. We provided an ERD, but you may want to add a separate Report table for the report functionality on the front page.
10. Authenticating users is very time consuming and difficult if new to the group. I would start it early and try to figure it out before the application becomes more in-depth. We waited until the very end, and it just seemed like an issue that would take many hours to implement correctly.
11. Refer to the GitHub Readme file as well just in-case anything is missing from this!