Purpose

This document was authored with the intent to provide a series of suggestions for subsequent CSCI-4250 (Software Engineering I) development teams regarding the ScavengeRUs project. Below, one may find section headers dedicated to general development tips and project-specific pointers to things we encountered or thought to pursue.

Working In Scrum

Kinser may be quick to mention that Scrum is a "Communist Love Fest," since no one person gets to dictate where the entire team is going at any one point. In our experience, the PO gets to set the vision of the development's course but is still subject to making sensible goals according to the developers' skillsets. **Generally, everyone will want to have a strong idea of who knows what, and at what times each group member is working/available**. Knowledge transfer is a fundamental part of collaboration.

Collaboration Spaces – Organizing the Group

Trello and Jira are the most immediately accessible collaboration frameworks. These spaces were made to, when aptly leveraged, allow development members to communicate their intentions and progress without *needing* a group meeting to do so. It will take some time to figure it out, which can be a good first step for the Scrum Master. These frameworks become much more relevant within SE2.

However, these frameworks aren't the end-all-be-all for SE1. We also used Discord to create text channels and breakout voice channels to discuss our findings as they came. In any case, it will be important to find whatever tools are necessary to keep the team talking beyond the in-class meetings.

Making Estimates

Estimating how long a *task* will take is in a very gray area. Experience will give you the best idea of what to expect but is further complicated by the condition of working with an unfamiliar codebase. Our group usually had at least one person who went an hour or multiple hours over the original estimate: the usual culprit was that in addressing the concerns of one task, a new problem would be discovered.

To navigate this, our group started to cast estimate votes (in Planning Poker style) to also include a 1-hour buffer for tasks that touched a new part of the codebase.

Project Points

Working with Views

One of the biggest nuances when working with webpage views is that their contents will not update to reflect realtime changes – sometimes it will be necessary to forcefully refresh the page to see updates.

Working with Docker Desktop

Docker Desktop is a great GUI for quickly publishing and testing containerized apps. Like any software, it has some nuances to it that may take some time to discover.

- An image is a "snapshot" of the project files at the time of image publication. Images usually do not change.
- A container is a deployed instance of an image. Each container instance is separated into its own runtime.
    - Containers may individually be booted up, paused, or shut down once created.
- Docker Volumes are individual directories *outside* of containers which can be mounted into containers. This technique allows for multiple containers to reference the same files (or create local copies and push back diffs).

Docker is intrinsically a command-line interface. Most online documentation will show off workflows using it. Furthermore, Docker CLI is (theoretically) required if using Docker to deploy to AWS. If you don't mind going ahead and looking into it, start now!

ScavengeRUs.db

A large part of the project depends on SQLite database file (ScavengeRUs/ScavengeRUs.db), which has a reputation for being a pain point within the project. However, there are some larger functional capabilities that should be worked towards if the team finds it a priority.

1. Separating from Project Files
    a. As of writing, all CRUD (Create-Read-Update-Delete) actions within the codebase expect the .db file to be inline with the base directory of the built app. This was a decent choice for initial development, but immediately becomes a massive inconvenience in efforts to deploy the app to a web server.
        i. The server's "filepath" is set up within appsettings.json file, stored in the variable "DefaultConnection"
            1. "DefaultConnection": "Data Source=ScavengeRUS.db"
               //Filepath to the database
        ii. It needs to be separated since, when deployed, each individual instance of the app would manage *its own* instantiated database. We need a uniform data store that all concurrent instances would reference.
    b. See various "Docker" and "AWS" documents throughout documentation in ScavengeRUs and etsuDummy/Kinserpedia. Investigating the deployment process now will gear you for SE2.
2. CRUD Anomalies

   a. CASCADE DELETE
      i. When a User is generated, their record is placed in the AspNetUsers table. These Users are then referenced by other tables (AspNetUserRoles, at minimum).
      ii. If attempting to DELETE a User that is referenced by other tables without a CASCADE DELETE, this causes an error. Essentially, the tables that reference the deleted record end up with a "NULL" reference. CASCADE DELETE will also delete any references to the deleted User record – all good! The tricky part is to make sure that all instances of touching the database appropriately CASCADE their transactions where necessary.
3. View: Error Page
   a. The current project (in development mode) will generate a page that details any unhandled exceptions. For database-bound errors, it would be nice to have a redirect to an error page reflecting such (e.g., "Sorry, this isn't available right now!"). It'd be a good day to find out the best place to implement such functionality (may be necessary to encapsulate each DB touch in try-catch, but *eugh*).

Deploying with Multiplayer Capabilities

A huge roadblock that our group encountered was in getting the deployed project to allow concurrent connections and to replicate interactions across clients.

Location-based Services (LBS)

Our group was unable to get a detailed look at enables LBS capabilities, but the *theory* is that the biggest hurdle would be to collect the device's location data from the running browser (if services enabled on host devices). This could be compared against the existent data within the Locations table in the database.