The database is a SQLite database.

It is stored as a file within the repository and thus using a git ignore is necessary to avoid merge conflicts between collaborators.

The database is a bit of an issue for source control because it is updated individually for each user due to it being a file.

The tables are based on the models in the code and when changes are made to the model a migration is made to update the database.

(in package manager console run "add-migration {migration-name-here}" "update-database"

An effective way to browse the database is DB browser for SQLite.

https://sqlitebrowser.org/dl/

An overhaul of how the database works will probably be needed in the future. One idea we had was to use data base scripts to populate the database each time the application was run. These scripts would be stored in the github repository. This would ensure everyone would have access to the same data. However, the way the database migrations work with it being linked to the code would probably make it tricky and we did not have time to put much effort into it. There is probably a better method than this that integrates more with the Microsoft system.

Kinser said that he would have preferred it if the earlier groups just used text files because the database is not really that complex. However, the solution already is built around the SQLite database, so it will probably be a good amount of effort to completely redo it.

Source control can be your best friend or your biggest obstacle. We want you to be able to hit the ground running, and minimize the issues that you will face with source control. We recommend that you follow these rules for managing your own git changes:

When you start the project, immediately branch off of main and create a development branch. Don't touch main until your final release at the end of the project when you merge development back into main.

When you start a new sprint, create a branch off of development for it and name it something like "sprint1" or "sprint_1".

When you set out to complete a task in a sprint, create a branch for that task off of whichever sprint branch that task belongs to on your backlog. Name it as descriptively as you can, i.e. "UserPhNumberValidationFix" or "AccessCodeNullRefFix".

When that task is completed and your code is tested, commit your changes and create a pull request to have those changes merged into the sprint branch. Get as many eyes on that code as possible and get two or more team members to approve your changes. After that, merge the pull request. Remember to delete the task branch after it is merged

At the end of every sprint, merge the sprint branch into development. You can either delete the sprint branch after, or keep it dangling for records-keeping purposes. It is a matter of preference.

The database file causes issues with source control because any time you run the program and use any action that modifies the database, git will detect those changes and try to track them. The database file is a binary file (as opposed to plaintext) and cannot be compared to an old version line-by line (diffed) however. This causes merge conflicts when you try to merge your changes to any code and there happen to also be changes to your local copy of the database. It is for this reason that we added a few lines to our gitignore to exclude the database from source control.

Possible fixes for this: Kinser suggested that we continue ignoring the database, but also make a few scripts that do the following:

If it does not already exist, create the database file.

If it does exist, drop all tables.

Populate it with the data that you want it to have. It will have this data for everyone.

Track these scripts with source control.

These scripts will be run upon every git pull. This accomplishes two things:

The database will no longer cause merge conflicts because it is not tracked by git

The contents of the database will effectively be plaintext, and therefore diffable.