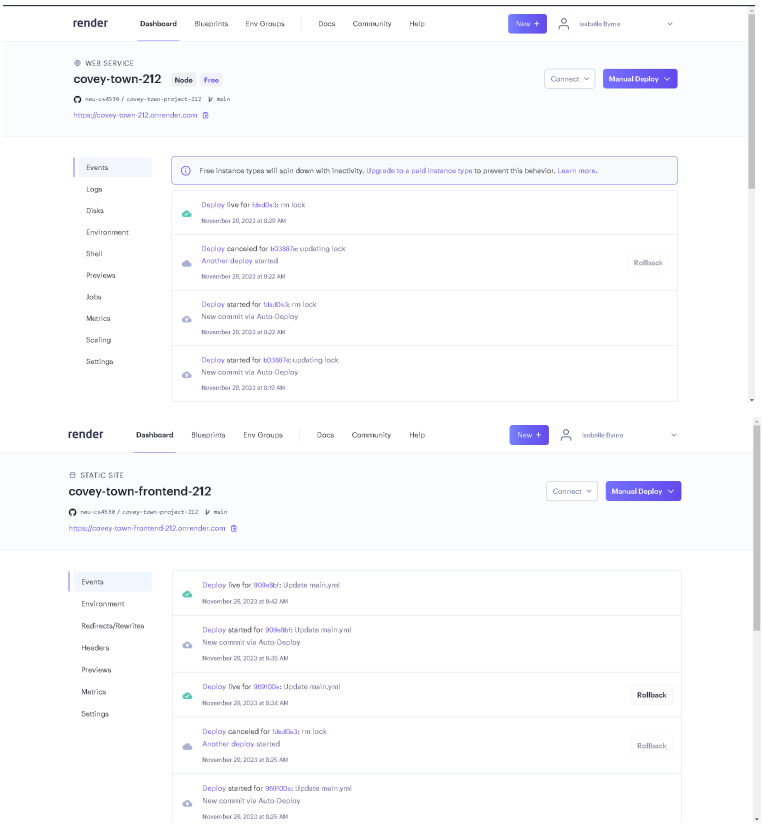
# Final Project Report

Christopher Lackowski
Isabelle Byrne
Jared Schmitt
Jennifer Shishmanian

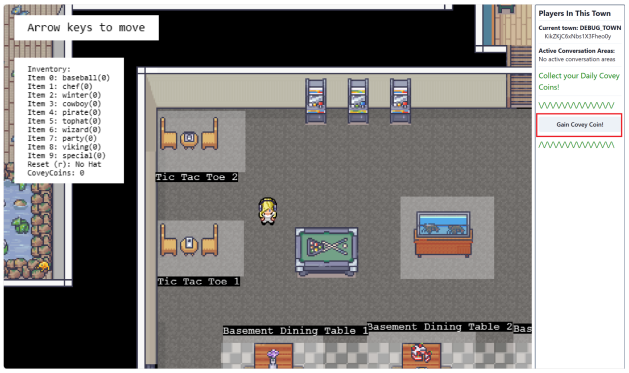
## Feature Overview/User Manual

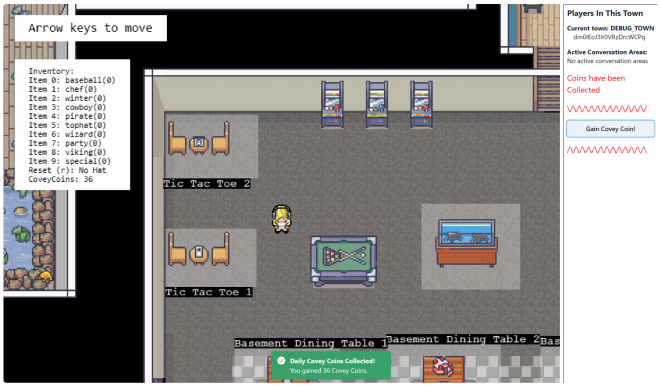
The following steps outline how someone, who is testing our project, should interact with CoveyTown and set up their world. These steps should be followed exactly because they will show off how our group successfully implemented all of the features that we promised to deliver in our 3 user stories. Throughout our development process, we were able to add features that allow players to collect Covey Coins every 24 hours, add hats to the character models to wear, purchase hats in a shop and trade hats with other players. Below are the steps that should be taken to test all of our new features in CoveyTown.

1) Press the following link to open CoveyTown and create a player with a memorable username to join the world(Assuming Render and Heroku's website/servers are working): https://covey-town-frontend-212.onrender.com/

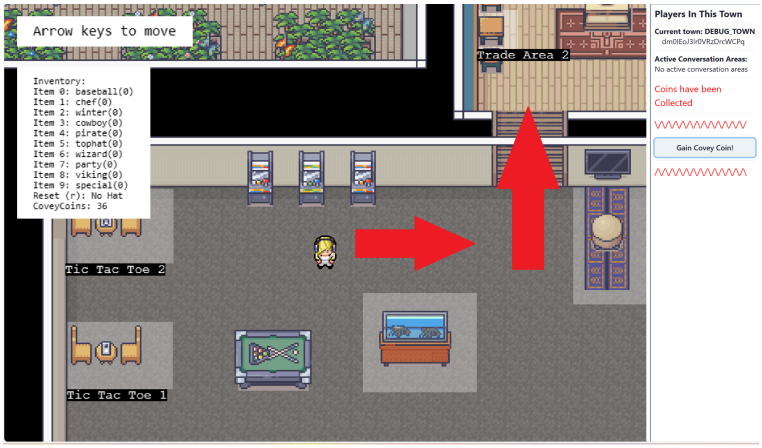


2) Press the button on the rightmost side of the screen that says ‘Gain Covey Coins!” to collect a random number of Covey Coins between 20 and 100. This number will be displayed in your inventory on the right side of the screen and toast will pop up on the bottom of the screen.

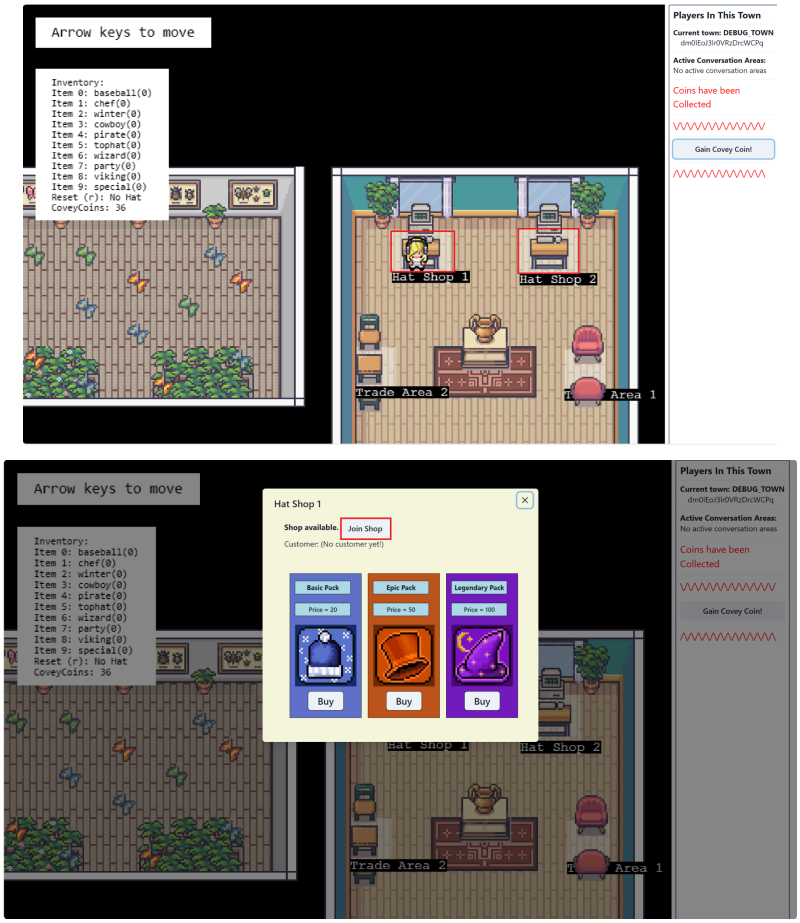


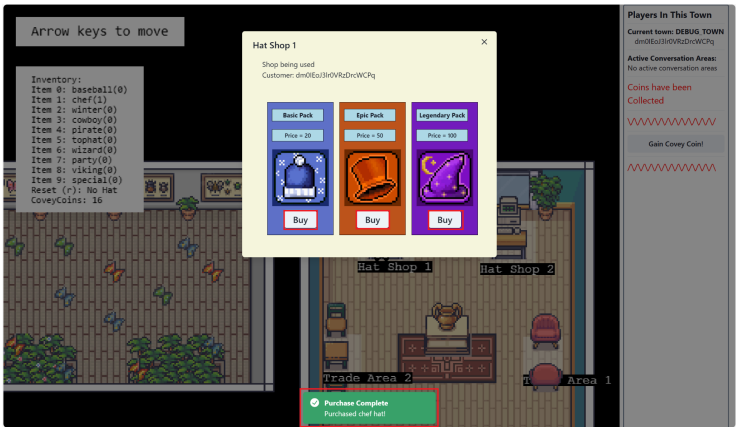

3) Walk up and to the right until you find a new area for everything related to hats.



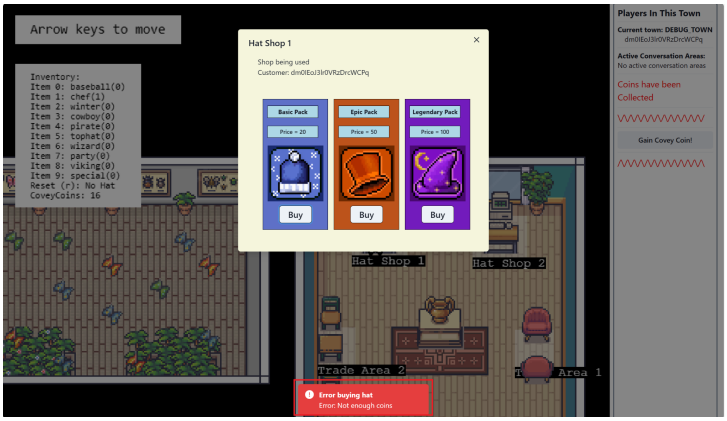
4) Walk to a Hat Shop and press the “Join Shop” button to start using the shop.




5) Buy any hat pack by pressing the “Buy” buttons on the screen. A toast should pop up telling you which hat you bought. There are random chances of getting 10 different hats across the 3 different types of packs that can be bought.

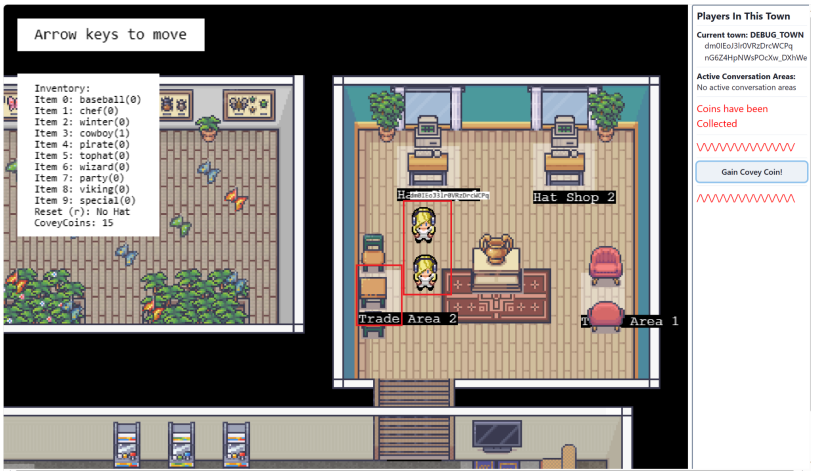


6) Keep buying hats until you have less than 20 coins and then try buying another hat to show the toast that says you do not have enough coins to purchase.



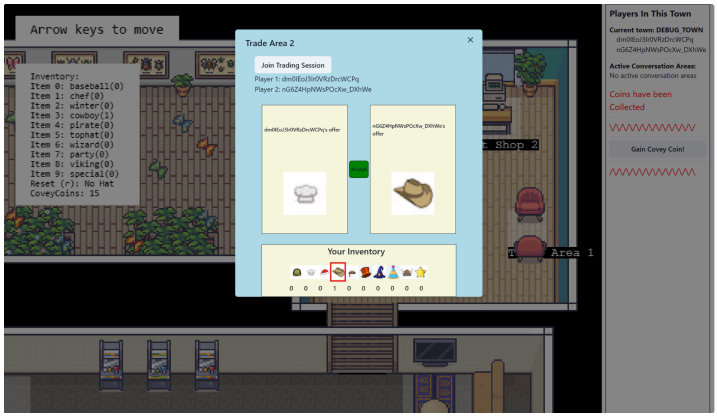

7) After leaving the Shop Area, the player can use buttons 0 through 9 on their keyboard to put a hat on their head. R can be pressed at any time to take the hat off the player and display the original character model.


8) Repeat steps 1 through 5 so that there are at least 2 players in the world with at least 1 hat each.

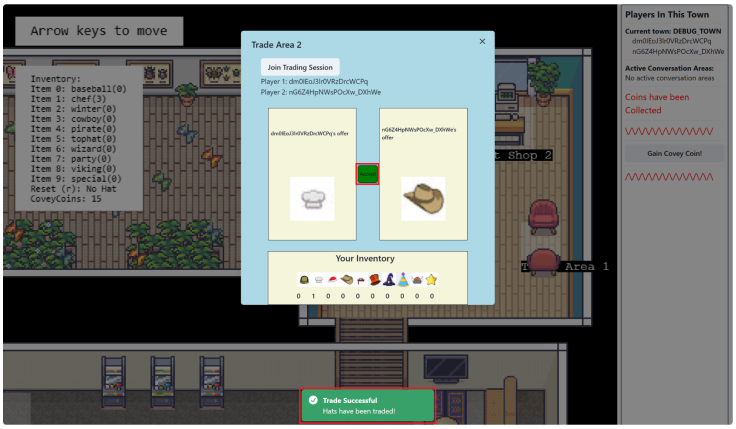


9) Both players should then walk to a Trade Area and join by pressing the “Join Trade” button.


10) The first player should press a hat button in their inventory to select which hat they want to give away and trade. Then the second player should do the same. Note that the numbers below the inventory hat icons tells the quantity that each player possesses.




11) Once both players have selected a hat to offer the other, either player can press the accept button to perform the trade. After leaving the Trade Area, the players inventories should be updated.



12) Lastly, close the world with either player and rejoin with the same username to check that the inventories of each player have been saved in a database and redisplayed.


## Technical Overview

In order to develop new features in CoveyTown we created many new directories, files and images and integrated them into the already existing codebase using Github. Generally, the overall structure of the code base remained the same and we made sure to abstract as much as possible to not interfere with already existing structures. Below we will discuss modifications and additions to the TownServices, Shared and Frontend directories.

TownServices:

To start, an API was created in the backend using mssql library to connect to a Microsoft Azure mySQL database. The design was created in a way that there was a separate class for all the query calls in order to make the API more readable.

Next, the trading and shop areas were created using the abstractions given to us by the creators of Covey.town. Each area has its own class for the implementation of area functions (functionality of the area e.g. trading area has trading functions and hat shop area has shop functions), a class for handling emitters and updating states of the area, a factory for initializing the area into the world, and a class for extending interactable to make the area an interactable.

Shared:

The only file that we modified in this folder was the CoveyTownSocket.d.ts file. This file is important for initializing and declaring many useful types that need to be used in both the frontend and townServices. So, we added many new types to this file to keep track of the names of hats, trade/shop states, new interactable types and the commands that can be emitted from interacting with our new areas.

Frontend:

In order to display visuals for the Shops, Trading Areas, Inventory and Coin Collector, modifications to the codebase needed to be made. Since we wanted the Shop and Trade Areas to

be intractable areas that the user can walk up to, press space and interact with a visual pop up, controllers needed to be created. The controllers that were designed and added to the code base allow the user's interactions with the frontend to trigger event emissions to the townServices which update the state of the areas. These files are extremely important in allowing the areas' information to be stored. The visual interactable pop ups were created as .tsx files. These files utilize chakra UI components such as containers, images, buttons and labels to allow the user to interact with the shops and trading areas.

There were also some other files that needed to be updated such as TownController.ts, TownGameScene.ts and TownMap.tsx. These files generally are used for setting up the frontend of the world and parsing the indoors.json file to know which interactable areas should be added to the world. So, there were many changes that needed to be made to ensure that our new areas would show up.

The SocialSideBar.tsx component was heavily modified in order to add a button that adds Covey Coins to the player's wallet if they press on it. They can only do this once a day. This was done because it looked the nicest to put the button alongside the other sidebar content.

The PlayerData.ts class was added to the frontend so that the group members who were not familiar with how the API worked had easy-to-use functions that would give them the information they needed. This made it so the group member in charge of the database did not need to explain every detail of how an API call works from the frontend. It made coding more efficient.

Since we needed to be able to see the new interactable areas for the Shop and Trade Areas, the indoors.json file was updated. This creates new interactable areas on the map that are later parsed and generated in the world upon startup. Additionally, all of the images for the hats and hat icons in the Shop and Trade Areas were custom created to match the style of art already in the world. These images were added to a hat-icon directory in the public assets folders.

To display the items that the player has in their inventory, we had to update the TownGameScene.ts so that this is shown at all times on the user's screen. Through calls in controllers upon button presses and other interactions, this updates whenever necessary to show the user the items they own.

We generally believe that we made the right design choices when deciding how to implement all of our desired features. We coded using the abstractions given to us by the CoveyTown creators and mostly added code when we had to change a pre-existing file. We did not interfere with the initial structure of the code and found ways to just add code where necessary. In the future if other developers were to work on this project further, they would very easily be able to create other types of shops and trading areas because of the ways we abstracted. So, we believe that we made the best possible design decisions.

## Process Overview

Our team communicated often and effectively to ensure that every task was getting completed in a reasonable time.

Sprint zero was carefully crafted so that we would have a solid plan going forward. An abundant amount of care and energy was put into making sure that we knew what we didn't know. What I mean by that is, we made sure to write down the code we had to write and the concepts we had to research in order to know what we needed to do. The idea in this sprint was to have a list of tasks that, if completed, would mean we were done with the project.

Sprint one occurred during the same week that we had our midterm for this class, thus we were unable to meet our initial expectations and complete the implementation phase of the project. However despite this setback, we cleaned up some of our plans and reconfigured our timeline. We all resolved ourselves to make up for it in the next sprint.

With efficient splitting of work, we accomplished an ample amount for sprint two. Our modified plan set everything back a week, but we realized that testing would not take as long as we initially had planned so the timings worked out. The weekly meetings with the TA recentered our focus on the tasks that were most important, and the frequent communication between team members maintained our ability to work in parallel. Despite our best efforts, we still had a small amount of implementation to finish due to an oversight of how much work our features actually take. The database also had many unforeseen complications.

In sprint three, we worked on finishing up the implementation of the features, and polishing the look of all the visual components. Up until this point, code had been written with the assumption that database code was correct. A small amount of time in this sprint was also spent connecting the database to the existing code. Due to an unforeseen medical emergency, Jennifer was unable to bugfix and merge her implementation of the coin collector area, thus we improvised and created a daily coin collector button on the social sidebar. After the final art assets were added we tested the final product and prepared for our presentation.

Throughout our work on the project, we utilized agile development in its entirety. By splitting the work into different sprints and completing our weekly retrospectives as a group, there was a lot of reflection done allowing us to adjust our work processes for the following weeks. Additionally, thorough code reviews before pull requests helped with catching small errors in the code and improving our skills as software development team members. Overall, we believe that we worked as a solid team and ensured that ample progress was consistently being made to create a solid final product.