# CS4530 Group 404: Final Report

*Harris Bubalo, Peter Li, Anna Murray, and Megan Nguyen*

## • Feature Overview

The following section details
- How to build and run our project locally
- Documentation and example use cases for our project

### Local Setup

To begin, access our codebase at https://github.com/neu-cs4530/spring-23-team-404 and clone the repository to your local machine. Because our project does not introduce any new technologies to the Covey.Town codebase, the setup is essentially identical to the starter-code setup:

#### ***Configuring the Backend***

To run the backend, you will need a Twilio account. To create an account and configure your local environment:
- Go to Twilio and create an account. You do not need to provide a credit card to create a trial account.
- Create an API key and secret (select "API Keys" on the left under "Settings")
- Create a .env file in the townService directory, setting the values as follows:

| Config Value | Description |
|---|---|
| TWILIO_ACCOUNT_SID | Visible on your twilio account dashboard. |
| TWILIO_API_KEY_SID | The SID of the new API key you created. |
| TWILIO_API_KEY_SECRET | The secret for the API key you created. |
| TWILIO_API_AUTH_TOKEN | Visible on your twilio account dashboard. |

#### ***Starting the backend***

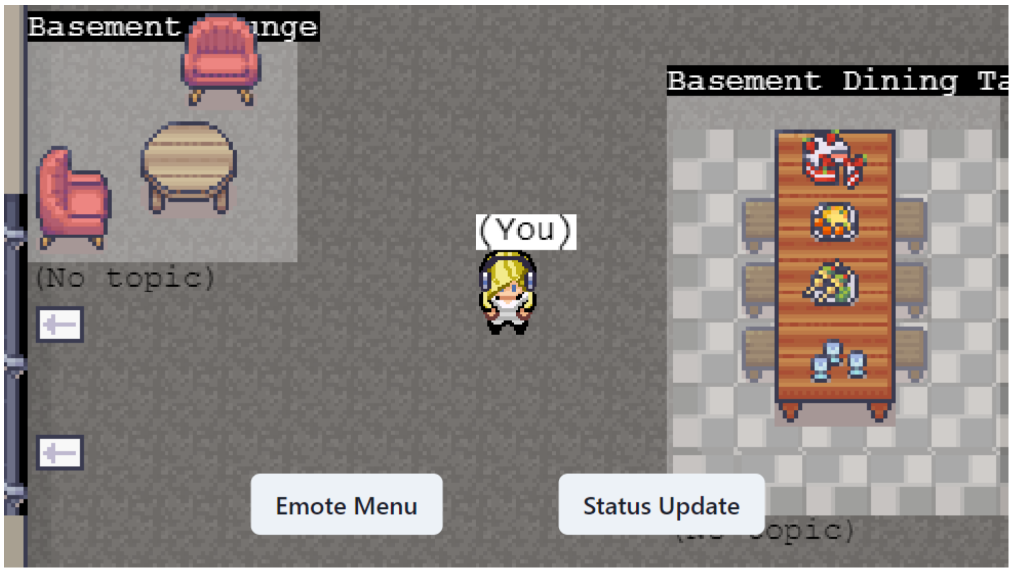
Once your backend is configured, you can start it by running “npm start” in the townService directory (the first time you run it, you will also need to run “npm install”). The backend will automatically restart if you change any of the files in the townService/src directory.

***Configuring the frontend***
Create a .env file in the frontend directory, with the line:
REACT_APP_TOWNS_SERVICE_URL=http://localhost:8081
(if you deploy the towns service to another location, put that location here instead)

***Running the frontend***
In the frontend directory, run "npm start" (again, you'll need to run "npm install" the very first time). After several moments (or minutes, depending on the speed of your machine), a browser will open with the frontend running locally. The frontend will automatically re-compile and reload in your browser if you change any files in the frontend/src directory.

## Documentation + Usage

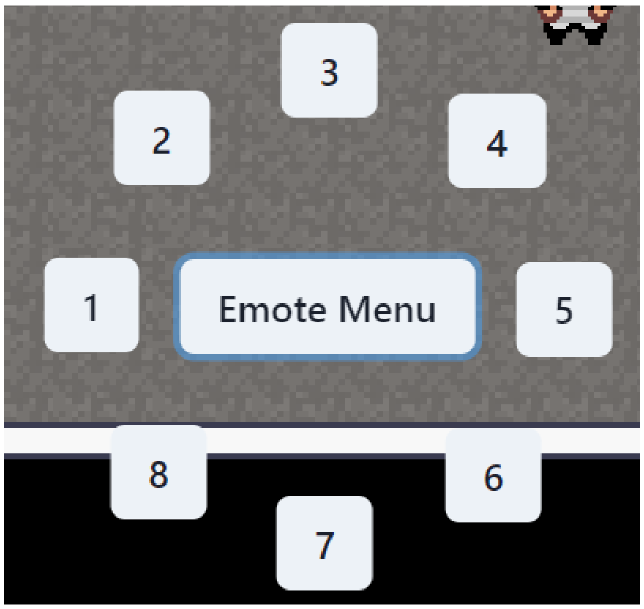
Upon entering Covey.Town, users are greeted to two new menu buttons within the game screen: the "Emote Menu" button and the "Status Update" button.



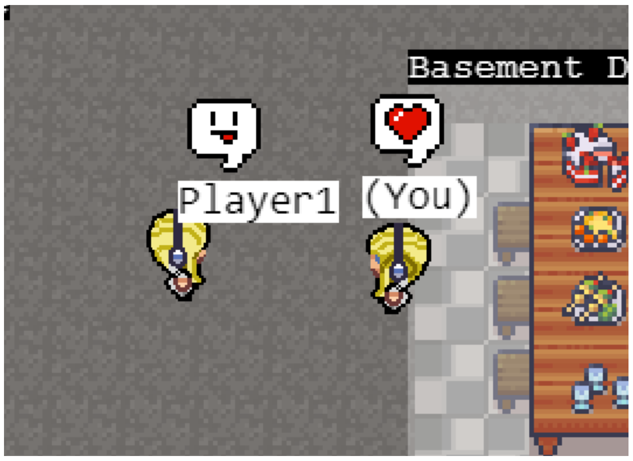
The Emote Menu pertains to our User Story 1 (essential), where we described providing users a way to visually express their emotions in the form of displayed sprites. The Status Update functionality pertains to our User Story 2 (desirable), where we described providing users a more constant way to express their emotions (as opposed to our short-term emotes) in the form of a visual "status", similar to the status functionality in something like Facebook (note that we decided to go for a text-based status instead of an image/emote based status in order to be more distinct from our emote functionality/User Story 1). Our User Story 3 was an extension task that we did not end up completing for this project (due to our focus on the other tasks).

***Emoting***

To display an emote, begin by opening the Emote Menu by clicking on the “Emote Menu” button. Doing so will open up a radial menu consisting of numbered emote IDs:

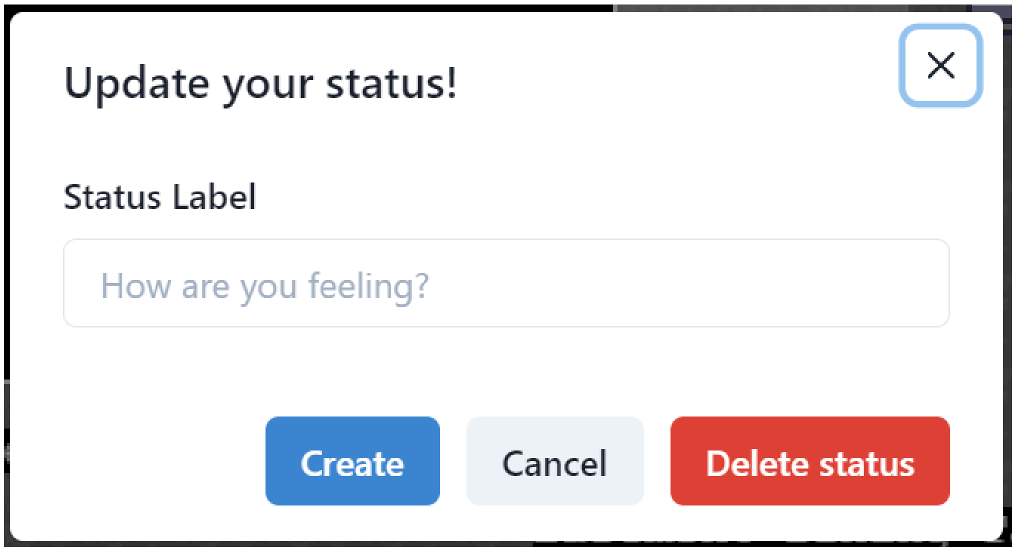


Each of these IDs corresponds to an emote. Selecting any of them will close the menu and display the emote sprite above the player’s head. The sprite will follow the player around as they move, and will disappear after five seconds. An existing sprite can be replaced if a player chooses a new sprite while a different sprite is active. Users within the same town will be able to see each other’s emotes.



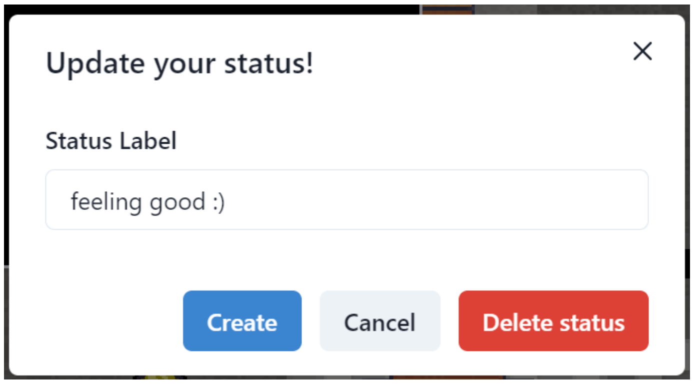
***Status Updates***

Users can also display a short (20 characters or less) tagline about how they are feeling underneath their player sprite. To do this, click on the “Status Update” button. Doing so will open up the Status Update modal.


Update your status!

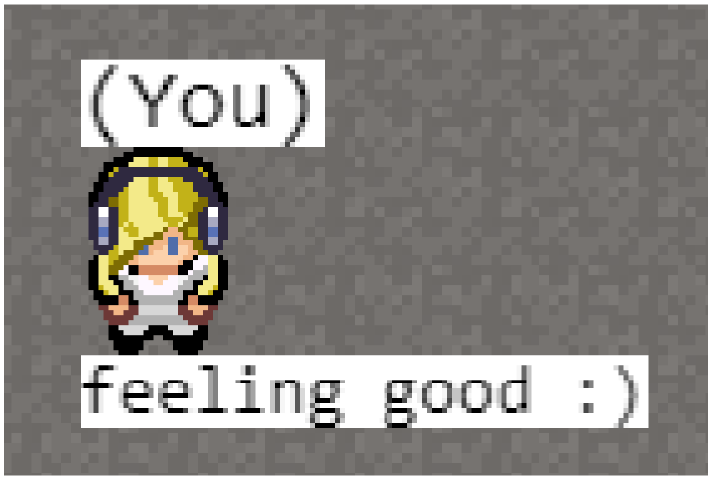
Status Label

Enter your status into the provided text box and click “Create” to see your status applied to your character. The status will remain displayed beneath your character until you either change it or click “Delete Status” from the Status Update modal. Like emotes, statuses are visible to all players within the same town.


Update your status!

Status Label

feeling good :)



## Technical Overview

The following section details

- The additions/changes we made to Covey.Town, and the design decisions involved with them
- Reasons as to why we made such changes + decisions

### ***Data Design***

In terms of our data design, we updated the Player class to include optional fields for *emote* and *status*. We did this so that TownGameScene can easily check when a player has an emote/status and render it accordingly (and then add the emote/status to the player’s gameObjects field). We made the fields optional since it is possible for a player to not want to display an emote/status, in which case the fields are undefined and TownGameScene does not render any emote/status. The status field is of type string, representing the text the player entered to be their status. The emote field is of a new type, Emote, which we designed as follows:

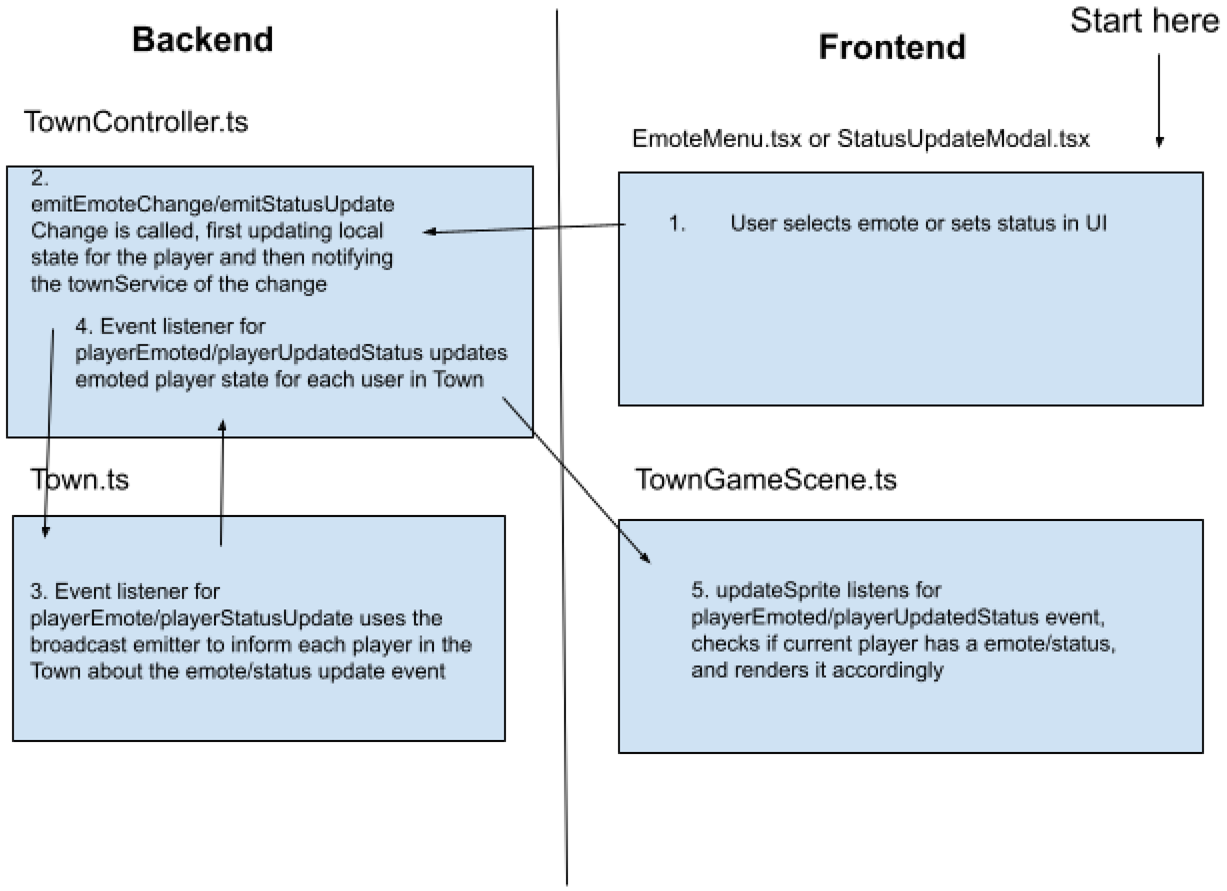
| Emote |
|---|
| id: number<br>timeCreated: Date |

We designed Emote in this way for two reasons: the number id field enables an easy way to match Emotes to a corresponding sprite in TownGameScene, since each of our sprites are named in the form “emote{id}.png”. Secondly, we needed to encapsulate the timeCreated information, since emotes should disappear after five seconds. By storing the Date at which the emote was created, TownGameScene can simply check (on each tick) if the current Date is greater than or equal to five seconds past the Date stored in the Emote, and then delete the emote accordingly.

### ***Architecture and Frontend/Backend Communication***

From an architectural perspective, both the emoting and status update events work similarly to the player movement event. This makes sense because, like player movement, emoting/status updates involve performing an action that results in changing the Player’s state (location vs. emote/status field of the Player) and emitting that change to all other players in the Town.

Below is a rough map of how the frontend and backend communicate to each other during emoting/status updates:


Backend
Frontend
Start here
TownController.ts
2. emitEmoteChange/emitStatusUpdate Change is called, first updating local state for the player and then notifying the townService of the change
4. Event listener for playerEmoted/playerUpdatedStatus updates emoted player state for each user in Town
EmoteMenu.tsx or StatusUpdateModal.tsx
1. User selects emote or sets status in UI
Town.ts
3. Event listener for playerEmote/playerStatusUpdate uses the broadcast emitter to inform each player in the Town about the emote/status update event
TownGameScene.ts
5. updateSprite listens for playerEmoted/playerUpdatedStatus event, checks if current player has a emote/status, and renders it accordingly

By using this architecture and workflow that is analogous to how player movement is performed, we make the codebase more consistent while still adding functionality. This poses a great advantage, as a consistent codebase is one that is easier to understand and build upon with more features.

## Process Overview

The following section details

- Agile project management techniques that we used throughout our process
- Expected vs Actual results for each sprint, and how we adjusted our approach as a result

***Our Agile Process***

We used a variety of agile techniques over the course of our project. In addition to our weekly TA meetings, we did a daily "stand-up" by texting our group chat with our progress, as well as thoughts on the project so far. For tasks that involved working together, we engaged in pair programming by hosting meetings over Zoom. In terms of code review, whenever any of us had a branch that we wanted to merge, we informed the rest of the group over text, and exchanged feedback either through texts or when we were able to work together in class. We also had something of a post-mortem when one of us accidentally installed a node package in the wrong directory, causing our Heroku build to fail; In this instance, we quickly diagnosed the problem together (as well as with the help of a TA on Piazza), and we respectfully discussed how to avoid this problem in the future (by making sure you install in the right directory). In terms of sprints, we followed the sprint structure as outlined in our individual project plan (with some variation in the expected results, see below). Lastly, we performed sprint reviews by working together on our sprint report every two weeks; As we worked on it over Google Docs, we discussed our expected vs actual progress, and how to change our approach going into the following sprint.

***Sprint Summaries***

<u>Sprint 0</u>

- **Expected Tasks:**
  - Phaser API research
  - Research as to how Covey.Town changes state on frontend
  - Rough Design for backend
  - Rough Design for frontend
- **Actual Results:**
  - All tasks described above
- **Changes to Approach?**
  - Too early to really say anything about our approach

<u>Sprint 1</u>

- **Expected Tasks:**
  - Design emote sprites
  - Code template for frontend changes

    - Code template for backend changes
    - Unit tests for backend changes
- **Actual Results:**
    - All tasks described above except template for frontend changes, as spring break made it hard to get everything done
- **Changes to Approach?**
    - Touch base with teammates more often to see when they are unable to do planned work, try to pick up some slack

<u>Sprint 2</u>

- **Expected Tasks:**
    - Implement frontend methods specified in template
    - Implement backend methods specified in template
    - Begin writing documentation (to be used in poster/report)
    - Write unit tests for frontend changes
- **Actual Results:**
    - Because of the complication in last sprint, we ended up postponing the unit tests to the last sprint, and replacing that task with template task from last sprint
- **Changes to Approach?**
    - Due to the slower progress than anticipated, we became a bit more conservative with our expectations and determined that we would not get to our extension task (User Story 3)

<u>Sprint 3</u>

- **Expected Tasks:**
    - Design additional emote sprites
    - Frontend changes for status update functionality
    - Backend changes for status update functionality
    - Tests for status update functionality
- **Actual Results:**
    - All tasks described above, except designing additional emote sprites, as we had decided that we would not accomplish our extension task
- **Changes to Approach?**
    - N/A, as this is the last sprint