Nathan White
Aron Kageza
Hanquing Guo
20180424
Rev 1.0

CS690 Buzzwords System Manual:

Introduction:

CS690 Buzzwords is a digital emulation of the game Taboo. It makes use of a highly decoupled frontend/backend architecture and leverages the strengths of websockets to provide a real-time, multi-client, user experience. It has been implemented in Python3 using the Flask framework for backend functionality and ECS6 with the Angualr1 framework for the frontend functionality. UI functionality, style, and animations were provided by the excellent Angular material design library in conjunction with angular-animate.

Hardware & System Requirements:

The application has been implemented as a web application, consequently, a network connected server is necessary. The server should have at least 1 CPU and 4GB RAM. The system requires a linux based operating system, but should be relatively straight forward to adapt to a windows environment.

Functionality

The application presents first time users with a simple dialog and asks them to log in, establishing a user name and password. On the menu screen, a user can decide to join an existing game that shows up in the menu, or create a new game. Once a user has joined/created a game, and a game reaches a valid state, the game flow will begin and each player will be selected for a particular role in a given turn. Each turn, a new team is selected to be "on deck" and a player from that team is selected to be the teller. A player from one of the other team is selected to be a moderator and all other players are assigned as observers or guessers depending on the team that is "on deck" or the turn modifier that has been assigned for a particular turn. Once a certain number of points has been achieved by a team, the game will conclude, showing scores and allowing players to return to the main menu.

How to use this document:

The following document explains aspects of the setup and use of our program. The following

document should not be consumed all at once, but referenced based on intended use.

The Project:

The following section describes the project development, the tools, subsystems, languages, and environments that were used in the development of the project. It also includes some architectural information in the section called Subsystems.

Subsystems:

Frontend:

The frontend subsystem includes all of the UI components of the application.

Backend:

The backend subsystem includes the database, game model, and necessary communications software to implement the interface with game clients.

Languages and Tools:

Languages:

- Python3
- Javascript - ECS6

Frameworks:

- Angular1
- Flask/flask-socketio

Key Modules:

- Flask-socketIO (backend)
- Socketio-client (backend)
- CORS (backend)
- socket.io-client
- angular-socket-io
- winwheel
- angular-ui-router
- angular-local-storage

Text Editor:

- Atom

Databases:

With regard to databases, the project has been configured to be very flexible. When running in dev mode a sqlite database will be created with a pre-generated schema in place in `./backend/db/` (assuming you are currently at project root). By default this database is called `dev.sqlite`. Details of this configuration can be seen in `./backend/settings.py`. For a production environment, a suggested MySQL configuration has been included in the file `production_settings.py` located in `./backend/`. Assuming the environment variable is properly set up and it points at this file, any settings which conflict with `settings.py` will cause the settings to be overridden with the production settings. Consequently, the MySQL will be used as the db for the applicationn. See `./backend/run` for the code that handles importing settings from the path pointed to by the environment variable. (TODO: implement this functionality)

Interfaces:

The frontend/backend subsystems make use of websockets for their communication. The websocket protocol enables persistent, full duplex communication between server and clients. This makes it simple to pass real time events from clients to the server and back again as is natural in a game environment.

For the development and deployment of the application we made use of python virtual environments. This makes it easy to ensure that your application's environment is both portable and stable.

Development Environment Setup:

In the root level of the project there should be a file called requirements.txt this contains the python packages that are necessary to get the backend up and running. To install the packages set up a virtual environment for the buzzwords project. To make this really slick, use virtualenv wrapper.

Initialize env:
(Assuming virtualenvwrapper)

```
$ mkvirtualenv buzzwords  --python=python3
$ cd ./buzzwords/
$ pip3 install -r requirements.txt
```

As part of this installation a package called nodeenv will be installed. This package makes it possible to virtualize the node environment inside of the virtualenv.
Add a nodeenv:

```
$ nodeenv -p
```

Then setup necessary packages:

```
$ pwd
.../buzzwords/
$ cd frontend
$ npm i
# wait for a while
$ bower i
# wait a while longer
```

Following the above procedure should setup the necessary environment for development work.

Running in Development Mode:

In the virtualenv for buzzwords, verify that run is executable, then:

```
$ pwd
.../backend/
./run
```

When this executes a new application will be started with the configuration specified in ./backend/settings.py.
Flask should spin up and a socketio server should now be listening on port 5000 (default port. see settings.py). At the time of the writing of this document there were no command line flags that could be passed to the ./run command, though it might be nice to be able to tweak certain aspects of the program's execution using flags in the future.

Production Environment Setup:

Frontend:

In order to set up the application for execution in a production environment, the front end files and all necessary resources should be available in some statically accessible web directory.

- Note: to get the frontend to properly reference the socket server, it is necessary to point the socketio-angular client at the backend. Consult the constant `socketIOConfig` in `frontend/app/scripts/app.js` to set the host and the port. This should probably be moved into some sort of a deployment procedure.

Backend:

A file called serve.py has been built for the purpose of running the server in a production environment. This will allow the use of gunicorn. From `./backend` run the following:

```
gunicorn --worker-class eventlet -w 1 serve:app
```

Alternately, if you want to get something up temporarily, you should still be able to use `./run` just be sure to update the ip address in `settings.py` and `production_settings.py` to the publicly availible IP of the host.

Running in Production:

The frontend app directory must be copied into a web accessible location. Make sure all necessary bower and node resources are also properly linked from within the web accessible directory. See index to figure out the path that the application expects.

Set up the backend to run as a process on nginx.