CPTS 423 Server Report

Team: SHAPE

Team Members: Patrick Bowling, Aaron Kwan, Andrew Lewis, Jonah Simon, Nick Strazis

Customer: Mona Ghandi

Old Server (Semester 1 Report)

For the back-end team, we were tasked with creating a way for grasshopper to take in the information sent from the front-end app and convert the information to something that can create the window.

We broke the task into three smaller more manageable tasks. The first being trying to find a way to connect grasshopper to the server itself. The second task then became communicating with said server so information can be transferred, and the third task was to then take the information and use it to create the window. The back-end team decided that the best way to create the backend application would be through grasshopper itself instead of a third party application. That way, any changes made to the application can be done with just grasshopper. As the back-end team worked on the project, we kept the front-end team updated so any problems could be dealt with immediately.

As we developed the backend for the project, we encountered a few problems that inhibited us from completing the project. The first problem we encountered was with the type of file that was being read in. Based on the file, processing the information would have to be done differently. In order to deal with this problem, the back-end team had to communicate with the front end app team in order decide on the file type that would be most convenient for both teams. After a few days of communication and testing a few file types, we decided that XML files would be the easiest to work with since there was a grasshopper plugin known as GHowl that could read XML files from a passed in URL. Another problem that the backend team encountered was that there was no way to efficiently update the grasshopper information after user input. At first, we attempted to create a messaging application that would send notifications to update after user input, but we encountered more problems trying to establish a reliable and secure connection with the server. We solved this problem by creating a timer that will renew the information every second, but this solution requires much more processing power as Rhino will have to redraw the window every second. We are still attempting to create the messaging application to make communication more efficient, but it is not complete. Other than these two problems, there were no other problems that are worth mentioning.

All in all, creation of the backend went relatively smooth and all problems encountered were dealt with. There are only two notable issues left; communication between the servers needs to be more efficient if we are to put Rhino and grasshopper on the server, and the backend application will need to be updated should the user want to put anything besides a location, a width, and a height.

New Server (Second Semester)

As we used our first server many issues became apparent. The old server was not able to handle rapid changes and would lock up. We did some research and learned that it was being caused by our server host. The host limited how many times a file could be changed in a 24 hour period. Since we did not have control of the server host, we decided we need to explore the option of creating a new server.

Research

We spent about a week researching and settled on the Django framework with the REST API extension and MySql to store the data. Django is a web framework for the programming language python and is used on many popular websites. We chose Django because of how it has been proven to handle a large amount of traffic. The wall is set up to ping the server every second and we have a large amount of input devices. For that reason, the framework we chose had to be able handle a lot of traffic. For our host, we chose pythonanywhere.com. Python Anywhere offers a free service for hosting Django websites. Additionally, they give full console access to the server. This was important to allow full access to enable the REST API extension.

Development

Creating the server on our local machine was very easy. The tutorials for the REST API extension were very simple and easy to follow. Problems arose when we worked on deploying the server. The first issue was that we could not clone our github because the privacy settings set by the school. To solve it, we had to create a zip of the server, upload it, then extract it using the server's bash console. The next issue occurred when we tried to access the server. At first, the server just displayed access errors. We did some research and found that the errors were caused by two things. First we had to update the settings.py file to contain the url of the server. Next, we had to create the MySql tables that we had on the local server. The final error appeared with the look of the website. All styling was missing. We learned we just had to add the path to the static files. Finally, our server was up and running.

Operation

The server interacts with all of the other devices through HTTP requests. Input devices such as Alexa and Kinect interact with the server by sending HTTP Post requests. The requests either go to the currentWall or the currentWallCoord endpoints. They are formatted as JSON Objects and contain the deviceID and the coordinates/configuration. The server then takes the configuration, deviceID, and current time and saves them in a MySql database. The wall interacts with the server through HTTP Get requests. When the server gets a Get request, it returns the most recent wall configuration as a JSON Object.

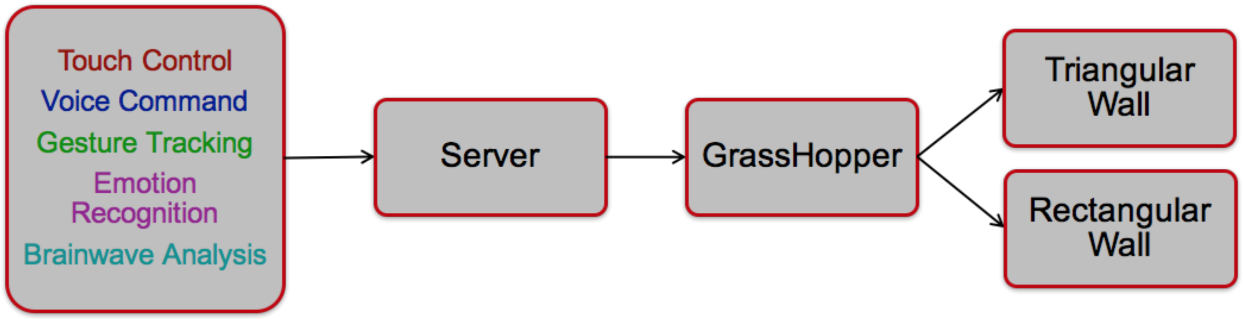
Machine Learning/Statistical Analysis

At the end of the semester, we experimented with implementing machine learning to the server. We started by improving data collection. We added the deviceID to identify the device making the change. Additionally, we spent a week populating the server with over 1000 changes/data points. Do to time constraints, we settled on performing statistical analysis. The analysis can be viewed on the mobile app. The server analyzes the data and sends it most common configuration given a time of day. For example, if the user uses happy in the morning the most, then the server will auto set the wall to happy in the morning.

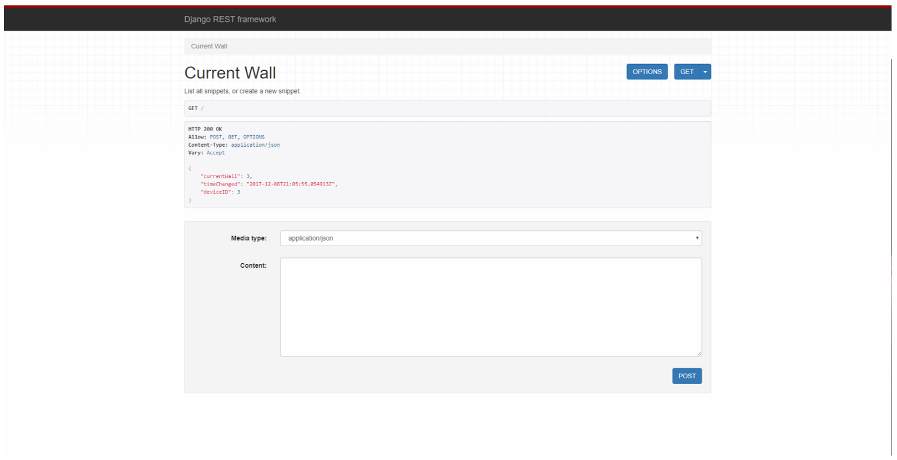
Links

https://www.pythonanywhere.com/
https://help.pythonanywhere.com/pages/FollowingTheDjangoTutorial
http://www.django-rest-framework.org/
https://www.djangoproject.com/
http://www.django-rest-framework.org/#tutorial

Appendix



Project Flow



Current Wall

List all snippets, or create a new snippet.

```
GET /
```

```
HTTP 200 OK
Allow: POST, GET, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "currentWall": 3,
    "timeChanged": "2017-12-08T21:05:55.054913Z",
    "deviceID": 3
}
```

Media type: application/json

Content:

Screenshot of the CurrentWall end point.