CS Senior Capstone Winter Term - Project Report

May 6, 2018

CDK Global: No more touch. No more Keyboard. Bring it All Together. Using Technology to Teach Humans.

Prepared for

CDK Global

Prepared by

Group 9

Look Boss, No Hands

Brandon Dring _____ Signature _____ Date

Nipun Bathini _____ Signature _____ Date

Carl Benson _____ Signature _____ Date

Abstract

With the engineering expo in sight, and the Spring 2018 semester, along with the overall Capstone Project wrapping up. This document goes over the current state of the project, what is left to do, any problems encountered and solutions to said problems. Lastly, it also includes snippets of code that are crucial to making the project work, and screenshots of the project in action.

CONTENTS

1 Purposes and Goals 2

2 Current Progress 2

3 Requirements Left 3

4 Problems 3

5 Interesting Code 4

5.1 Request Refinement 4

5.2 Suggestions 5

6 Images 6

1 Purposes and Goals

The purpose of our project is to develop a new way for users to interact and view enterprise data. This new method will replace traditional input mediums like the keyboard and mouse that are used to scroll and click through data, instead using new technology which will allow users to use their voice and hands in virtual reality to view graphs, charts and more. For the voice technology, we are using Amazon's Alexa, due to its friendly developer environment, along with robust and accurate voice recognition built in. From there, the parsed sentence is sent to our AWS server, pulling and parsing data, then displayed in Virtual Reality in the HTC Vive.

This project was designed for our client Trevor Moore and Deepak Paul, and the company they work at CDK Global. The plan is to place both the Amazon Alexa, and HTC Vive in a room, in which the dealer can walk in and begin using the custom Alexa skill. Using their voice they will be able to make a requests, for example asking sales data in a state. Alexa will then retrieve data from our database and present the data in the headset. In the virtual reality, our user will be placed in a virtual house, displaying data in various corners. The user is also given the option to look at a world map to zoom into a particular location to view dealerships.

2 Current Progress

From receiving the Alexa's near the end of fall term and having the entire winter and partial spring term to work with the voice assistant, we have met the goals laid out in the requirements document. During fall term, we had the opportunity to create basic Alexa demos, and get a brief understanding of developing with Alexa and AWS. Later we populated our database with fake sales data, which was created from our data generator. This allowed us to later connect Alexa to our MongoDB database where all the sales data is stored. A data model was defined for Alexa requests, the model contains a plan of what Alexa commands should be used and the capabilities of the system.

Tests were added during winter term, to ensure that voice commands output data on the web socket, and that Alexa is properly interpreting commands. When Alexa starts a VR environment, it stays open, or at the very least, remembers what the last state of the VR machine was. From there, Alexa is able to take commands and then filter down for example from a year to a month. We have now provided Alexa with a semi-intelligent response guide, making it sound more human with logical responses to any questions asked, instead of being limited on its responses, making it sound as if it was a hard coded robot. The voice assistant also contains code for error handling, so when the user makes an improper request, Alexa will be able to guide them to a valid request.

We have implemented graphs that map to the VR headset via Alexa commands, the user can now say a wake command and they will see the VR switch from a house to the world map package we have installed. Our data generator produced a .CSV file, which we parsed and retrieved approximately 18k cities and 55 states and islands. These city and state values were then added to the Alexa skill, so they could be used in commands. Using the testing configuration tab found in AWS we added multiple testing combinations to the tab using JSON requests, the tests all currently pass showing that our current Alexa commands are valid and working. In order to better enhance the user experience, before each request, using the userID, the last request that they had made is pulled. The data that isn't null

is used to populate the same fields in the current Alexa request before getting new data.

3 Requirements Left

There isn't a whole lot to do left for the project as all of our requirements have been met. When meeting with our client, they have also expressed that we have completed any extra features that they had wanted to see that weren't listed on our requirements document. So, we believe we are done with the project and can take this current version on to expo.

After expo, our client had expressed interest in continued development of the product at CDK. Trevor specifically is excited about the thought of using the Alexa to get data and analytics for CDK. So much so, that he will probably use the Alexa Skill for the intern project at CDK Global this summer up in Portland. However, the low fidelity of VR is sort of up in the air for further development until it is better supported.

As for our stretch goals go, we never got around to implementing a heart rate monitor for when the user is in the VR environment and when they are looking at and experience the project. Largely due to the fact that we didn't have the bandwidth to take on such a task, in the allotted time. Compounded by the fact that the turnaround time for getting a relatively expensive piece of hardware from CDK is lengthy.

There was another stretch goal, although I don't believe it was written in the requirements document, but mentioned verbally. Which was implementing real machine learning and artificial intelligence, such as Google's Tensorflow, or using one of the solution AWS provides to handle Alexa's speech responses and suggestions. Instead a rather rudimentary implementation of suggestions and responses were made. But, the project could still support them in the future with further development.

4 Problems

Making linear progress on the project has been a problem, there seems to be sprints of rapid development over the course of 1-2 days. Then, the project just sits idle not being worked on for the rest of the week. These code sprints are also typically right before larger client meetings, demo's, or meetings with our TA. Thus, the resulting work done for the task of the week can be sub-par, and lead to more work later on. Now that the project is done, all the sub-par work has been resolved and refactored, but something noticed throughout the project.

While the VR environment was implemented it was rather low fidelity. This isn't necessarily our fault, if you look at most games in VR, the graphics and pixel density leave something to be desired. Doubly so when using indie developer packages, and being implemented by students with no VR experience in the first place. As such, the quality of Virtual Reality looks cheesy for lack of a better word. It is quite noticeable and ruins the immersive experience, not exactly garnering the idea that it could realistically be used right now. There is no current solution for the lack of graphics as Virtual Reality is still in its infancy, maybe in a few years it will be up to where console game graphics are now.

Being able to demo at expo might be a problem, our project requires that we have rather fast internet, along with a large space, and a semi quiet environment. The expo has none of those things, along with a computer needed to run it, one of us will have to bring one of our personal computers to expo. For the interest of time, because it's already been installed and set up.

Getting our project to work as an executable for all to use is an issue. Currently, we have the project set up under development modes, such that it is tied to a developer account both on Amazon, and Unity. Getting the project out of both of those environments requires them to go through a lengthy review process by their respective companies. Instead, we will have to share the developer account information with CDK if they want to run it locally in their offices.

Unity provides its' own version of version control, that enables you to push and pull commits directly from the IDE, and it syncs with the team account perfectly. So we ended up using it, however, it is definitely not as user friendly, and functional as classic Git. This led to every commit going to the main branch, and resolving conflicts next to impossible. In the future, it would be helpful to look for a better version control system for a project that can be as complex as VR can be.

There was an issue with developing the Alexa skill within AWS as well. There wasn't great support for version control, as such each change that we made went straight to production. Meaning that during development often times it would be completely broken until all the bugs were for that small code sprint. Now that the project is in a stable state it's not an issue, but in the future, it would be nice to see how professionals develop Alexa skills.

5 Interesting Code

5.1 Request Refinement

One of the requirements set by our client was that a user can refine their previous search. For example, if their first search was for sales in Oregon, they should be able to search for Portland, Oregon by requesting just Portland. To achieve this, we added two additional functions to the AWS server portion of the system. The first is a request logger. In order to actually built upon the previous request, we need to know what that request was. To do this, we implemented a function that runs during the processing of every request. This function logs the user's ID and their request into the database.

```
let lastQuery = {
    userID: userID
};
_.isEmpty(req.body) ? null : lastQuery.body = req.body;
_.isEmpty(req.params) ? null : lastQuery.params = req.params;
_.isEmpty(req.query) ? null : lastQuery.query = req.query;

lastQuery = this.suggest.createSuggestion(lastQuery);
let inserted = await this.insert(lastQuery)
return inserted.ops[0]
```

Now that we have a record of the previous request, we can merge it with a new request to fill in any blanks. If the user wants to start a new request, that doesn't build upon the previous one they can include the keyword "new" in their request. If this keyword is used, the value of `req.query.reset` is set to true, which indicates to the function to simply return the current request unmodified. Otherwise, it retrieves the most recent request from the user and uses the built in `Object.assign` function to merge the two objects together. The order the two requests appear in the function call is very important. Having the current request come second means that any conflicting values between the two will default to the one in the current request. `Object.assign` does not work well with subvalues within objects, such as `req.params`, which makes multiple calls necessary. `req.params` contains the values of the request, such as location and grouping method. `req.query` contains information about the request, primarily the userID.

```
if (req.query.reset) {
    let user = req.query.userID;
    let lastRequest = await this.findLast(user);

    let mergedRoute = Object.assign({}, lastRequest[0], req);
    mergedRoute.params = Object.assign({}, lastRequest[0].params, req.params);
    mergedRoute.query = Object.assign({}, lastRequest[0].query, req.query);
    return mergedRoute;
}
else
    return req;
}
```

5.2 Suggestions

Creating a way for Alexa to suggest follow-up requests is problematic. To make meaningful suggestions would require machine learning of some sort. Rather than add another large requirement to an already sizeable project, we went with fairly simple suggestions. One suggestion Alexa might make is for a request one level of refinement up from the current request. For example, after a request for Portland, Oregon, Alexa might suggest a follow-up request for Oregon as a whole.

```
let suggestionQuery = JSON.parse(JSON.stringify(lastQuery))

if (suggestionQuery.params.name) {
    delete suggestionQuery.params.name
    return suggestionQuery
}
if (suggestionQuery.params.city) {
    delete suggestionQuery.params.city
    return suggestionQuery
}
```

6 IMAGES

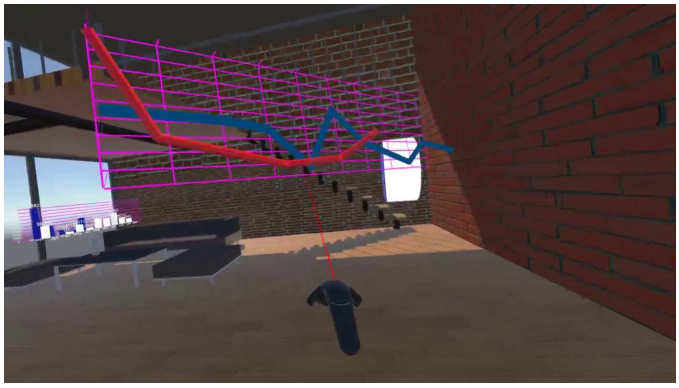

Fig. 1: A line graph showing the sale trends of two groups

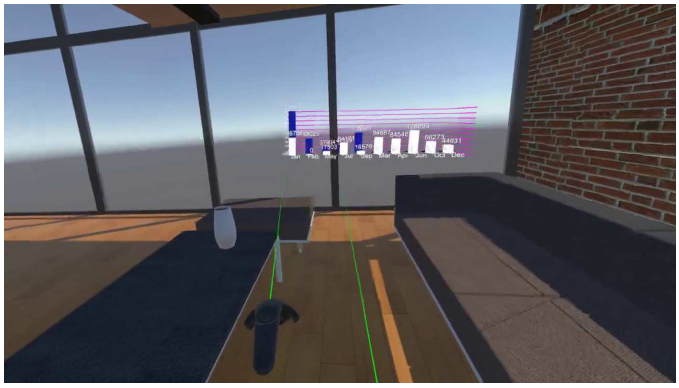

Fig. 2: A bar graph showing sales trends of two groups, broken up into month increments

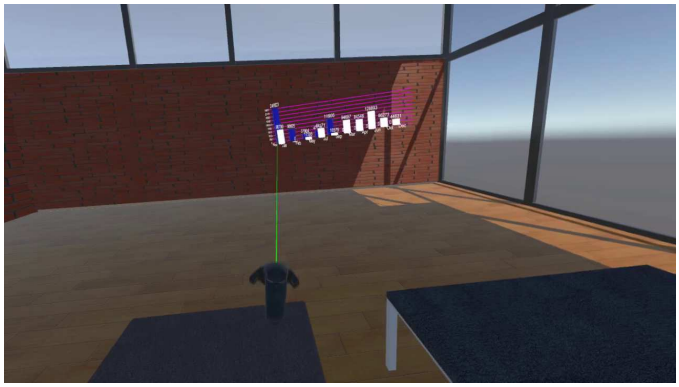

Fig. 3: A graph being selected and moved around, note the green beam indicating a selection