COM620 Immersive Technologies Assessment

[Document subtitle]

Paul Dibble

[Year]

Link to live application on git hub pages: https://pauld001.github.io/ImmersiveTechnologiesARPOI/

Introduction

In this assessment task I carryout assessment task B. Augmented or virtual reality points of interest application. For the project I focus on the augmented route. In the application I needed to produce a AR web application. The application would download points of interest from a web API. The application would pull POIs appropriately to the user’s location. This was either done by tiling or a bounding box. In the application is uses the binding box method. the application needed to display models for each individual POI. Differenced by poi type. E.g all restaurants would have a different model to all pubs. POI name in information would also be shown. Clicking this info should take the user to the website. the application would also need to display current longitude and latitude.

Software and libraries

To develop the application, I used the web application languages html, and JavaScript. with these languages I additionally used package libraries

. To install these packages, I use the node.js package manager.(<https://nodejs.org/en/>). Using ‘npm install packagename’ in the project directory to install packages into the project. Node package manager installs the packages for the user and downloads them into the project ready to be used. This would be an alternative to manually installing each package.

In the application is used the packages; a-frame, ar.js, jsfreemaplib, http server, and webpack.

a-frame is a “web framework for building virtual reality”(a-frame, GitHub, 2021 a-frame helps the user to create a AR and VR application in the browser. A-frame creates an environment for the application to run in and helps the user create 3D objects and import models for the application.

AR.js is an “open-source” library that’s used to integrate AR “functionality” in a “few lines of code” (EGINGTON, 2019)this library makes it easer for the user to integrate AR in to a html and JavaScript application.

Jsfreemaplib is a package package that contains is used to help work with “geodata” (npm, 2021) and is also used for “Mercator projection” which would be used to find the coordinates of the camera.

Node http-server a node package used to host a HTTP server locally on a device.

Webpack –

Code

the application consists of:

* index.html - the html based front end of the application.
* index.js- the JavaScript backend of the application.
* node\_modules - generated by node contains node and installed dependencies.
* file models – contains the 3d models used in the application.
* package.json- generated node or the user. List of decencies. installed in the application. Running npm install in project directory would install listed dependencies.
* Package-lock.json. generated by node. A package tree Used to” keep track of versions of packages” (contributors nodejs, n.d.) useful for package version control.
* Webpack.config.js- configuration for webpack.

Index.html

Html the front end of the application.

Text

Description automatically generated

Figure 1 creating the scene

In the figure above using the syntax a-frame“<a-scene>” I set up the scene of the application. I call the scene “POI” as I am making an AR application I set “vr-mode-ui” to false and set up ar.js setting the AR source type as webcam.

I additional set the raycaster in the scene. Used to set distance of mouse clicks unfortunately did not implement features using mouse clicks.

Text

Description automatically generated

Figure 2 models

In figure 2. I use a-frame to create the assets of the application. in this case the assets are the 3D models. The models I used were all downloaded as a gltf format. Each of the models I used I credit the author and link the original source of the model within the code. Using the syntax “a-asset-item” I define the source gltf file of the model e.g: models/map\_pointer/scene.gltf. also giving the asset an id.

Text

Description automatically generated

Figure 3

In Figure 3. I create a new div wrapper “infobox” which I position to the top left of the application above the a-scene. in this div I create a paragraph with the id of “info” which would be filled in from the backend with the user’s current longitude and latitude.

Index.js

the JavaScript backend of the application.

Text

Description automatically generated

Figure 4 importing dependencies

In figure 4 I used “import ‘*dependency*’” to import required dependencies to be used in the application.

Text

Description automatically generated

Figure 5

“Aframe.registerComponanent” is used to import the scene POI which I created in index.html.

Running the component initiates the function the application runs in.

A screenshot of a computer screen

Description automatically generated with medium confidence

Figure

watchPosition is used to update the geolocation every set time period to update to the user’s latest location. On update runs the async function.

“Document.getElementById(‘info”).innerhtml” is used to get the element “info” which was defined in index.html and set it to “`Current Location: Lat ${gpspos.coords.latitude} Lon ${gpspos.coords.longitude}`” with ${gpspos.coords.latitude}/ ${gpspos.coords.longitude}`” being the latitude and longitude from the geolocation updated by the watch position.

A screenshot of a computer screen

Description automatically generated with medium confidence

Figure

In figure 7, I create a web Mercator google projection. A projection “increases size of areas towards the equater” (Barcelona Field Studies Centre S.L., n.d.) creating a flat map with a modified latitude and longitude. This would be useful in the assessment to improve accuracy on the “flat 3d world” (class notes, Whitelegg, n.d.). This would allow the pointers to be located more accurately on a flat level.

In figure 7 I’m defining a new projection “GoogleProjection()” called “location” . Then creating the variables “lon” and “lat” from gpspos.cords.longitude and gpspos.cords.latitude. then using the function “…project(lon,lat). A projected easting and a northing are being created.

Text

Description automatically generated

Figure

In figure 8 I use the projected easting and northing to set the position “position” of the camera “[camera”].

Text

Description automatically generated

Figure

In figure 9 I create a bounding box. Setting the “longitude” to 0.05, and “latitude” 0.02. with requirements for the assessment. I chose to use a bounding box, instead of a tiling as I have more experience with it from the week’s lab classes. Although tiled would be useful for decreasing requests form the server.

Text

Description automatically generated

Figure

In figure 10, the location variables created in figure 9 “west”, “south”, “east”, “north”. Are sent off in the fetch request to the server using an API. This downloads the point of interests within the given coordinates as a json format.

Text

Description automatically generated

Figure

In figure 11, a “forEach” loop is used to run the function for each record. For each record new entities are created. Entities “point” and “text”, that become child classes of point text.

Text

Description automatically generated

Figure

As shown as figure 12, I used if and else if statements to cycle through each “amenity” and choose the appropriate model. For the assessment I sorted between cafes, restaurants, and pubs. Any location that had different type. Or value was null would be given a default model.

Text

Description automatically generated

Figure

In figure 13 I set the attributes of the models and the text: “scale”, “value” and “position” position was calculated using a google projection on the coordinates from the “api”

the classes are then appended to the scene.

Outcome

Up to a certain extent the application worked. First the application is successful able to download points of interest from the API

Graphical user interface, text, application, email

Description automatically generated

Figure application console

Figure 14 is a screenshot of console in the browser. using: “console.log(`name: ${feature.properties.name},${feature.geometry.coordinates}`);” I was successfully able to output name and coordinates of each POI. The application is successfully locating the devices coordinates, and with the bounding box sending the east,south,west,north to the API and getting a response back.