**Instruction Manual**

**for**

**360 Virtual Tour**

# For Users

## On PC

* Open any web browser and copy the link <website link>
* From the initial black screen, click the arrow pointer to enter the start location.
* Each location has a description button, a volume button, and labelled arrows.
* Clicking the description button will open up a panel that describes the current location.
* The “Toggle Audio” button turns on/off the audio playback of the description.
* The “Website” button redirects the current page to the location’s official website, in case of departments and landmarks.
* The background music can also be muted/unmuted by clicking the volume button located right next to the description button.
* The locations nearby are marked by the labelled arrows. Click on any of the arrows to move to the specified location.

## On Mobile (Note: Current support for Android)

* Open any web browser and copy the link <website link>
* From the initial black screen, hover over the arrow to enter the start location.
* Each location has a description button, a volume button, and labelled arrows.
* Hovering over the description button will open up a panel that describes the current location.
* The “Toggle Audio” button turns on/off the audio playback of the description.
* The “Website” button redirects the current page to the location’s official website, in case of departments and landmarks.
* The background music can also be muted/unmuted by hovering over the volume button located right next to the description button.
* The locations nearby are marked by the labelled arrows. Hover over any of the arrows to move to the specified location.
* \*Hovering in above instructions assumes a hover duration of 0.5 seconds.

# For Developers

Clone the frontend and backend repositories from the github website.

## Frontend

* Use npm install to install required packages.
  + Front end requires vite to build and run.
* The index.js file contains the main static website structure.
* The public folder is the place where all assets that are directly to be used by the static website are stored. These assets are available instantly once the website loads.
* The js folder contains the script.js file which enables the site to function.
* Make sure to create a dist folder if not already present in the clone. This folder is necessary for the build to be stored when the website is deployed.
* The css folder is currently empty. But any additions required to style the commands in the website can be added here.
* Note that during deployment if you change the backend website details you have to change the same in the code as well. This change has to be made in the variables available at the top of the script.js file which are backendWebsiteURL and cloudfrontURL.

### About Frontend code

* In index.html
  + Information regarding a-frame primitives and API can be found at <https://aframe.io/docs/>
* In script.js, custom components are registered and given functionality as described below
  + ‘spot’ component contains information regarding the current location and neighbouring locations. It is responsible for updating the scene image, adjacent arrow names and directions, description panel values, and certain audio playbacks. It acts as a manager component.
  + ‘lookatcam’ is a trivial component that adds the look-at=”#cam” property to the element it is attached to.\
  + ‘audioplayer’ is a component that toggles between play/stop states of the description audio.
  + ‘openbox’ is a component that handles opening/closing the description panel, placement of description data, and switching between background audio and description audio.
  + ‘noclick’ is a component that makes the element it is attached to non-interactable by the cursor’s raycaster.
  + ‘soundreset’ is a component that sets the state of description audio to stopped, once it has finished playing so that it can be replayed by ‘audioplayer’ component.
  + ‘togglebgm’ is a component attached to the volume icon button that mutes/unmutes the background audio.
* Additionally, the cursor click method is changed depending on whether the device is a PC or a mobile. This is done before registering the other components.

## Backend

* Use npm install to install all packages required for local running of the backend.
* Use “npm run start” for running “node server”
* Use “npm run dev” for running “nodemon server”
* Create a .env file in the root folder and refer to the environmentrequirements.txt file for the environment variables that are required to make the site functional.
* The server.js file is the main express server file which acts as a gateway to all functionalities of the backend.
  + It connects the site to the database and redirects some routes to the appropriate routes in the routes folder
* The views folder contains the html files that act as the content manager interface for the backend.
  + contentManagerinterface.html file is the file that acts as a menu that redirects you to the right form for the determined action. It can be accessed by <backendWebsite>/contentManager
  + dataFill.html is used to upload new data to be stored.
  + deleteData.html is used to delete unwanted location data.
  + index.html is currently used as an empty entry portal if you enter just the backend website address in the browser.
  + getData.html file make sure to insert the right backend website for getting all database information.
  + patchData.html is used to update data in the database. You can enter all information you want to change in the database but the only argument necessary is the building name. Make sure to match the name with the one you’re trying to change.
  + patchImage.html is to update the image of the required building.
  + 404.html is the site redirected to when an unregulated website path is entered in the browser.
* Routes folder contains various routing javascript files.
  + images.js routes information entered in the forms to the mongodb database.
  + interface.js routes contentManagerInterface page links to the appropriate sites.
  + root.js is just to route the main ‘/’ site to the index.html site
  + s3.js is to make sense of all form submissions and perform changes in s3 as well as redirect information to the image route.
* Public folder contains a style.css that updates the index.html page now. More styles can be added to it if necessary.
* Models folder contains an image.js file that creates the schema for the mongodb json requests that are sent. This is used directly in the images.js route
* Middleware folder contains middleware that provides additional information regarding functionality to the user.
  + errorHandler.js routes error messages to errLog.log file
  + logger.js logs backend requests to a log file in the logs folder.
* The config folder contains configuration setting files
  + allowedOrigins.js sets origins which are allowed to send requests to the backend api
  + corsOptions.js sets Cross Origin Resource Sharing Options and checks whether the request is sent from an origin which is allowed by the backend.
  + dbConn.js is used to connect to the mongodb database using the URI.

### MongoDB Atlas setup

* If you’re running another account to test out, then create a mongodb atlas account and create a collection and database. And use the cmdline tools instruction in the website to get the URI to be pasted into the environment variables.
  + Make sure to name the database name alone as “images”
  + URI general format: mongodb+srv://<username>:<password>@virtualtourcluster.j3xikw6.mongodb.net/<CollectionName>?retryWrites=true&w=majority
  + Also in the networking options add user to access from any origin / ip i.e. 0.0.0.0/0

### S3 Setup

* If you’re creating another aws account to run this project, then once you create the account go to the s3 page and create a bucket.
* You have to create an IAM user and create the user policy to point towards the created S3 bucket.
* Once the IAM user is created, create a security credential - SECRET\_ACCESS\_KEY and ACCESS\_KEY which are to be entered in the .env file.
* If any doubts occur use this video for reference - [Reference](https://youtu.be/NZElg91l_ms?t=276)
* Update S3 permissions. Scroll down to update CORS policy to the following - refer this aws link for the policy - <https://docs.aws.amazon.com/AmazonS3/latest/userguide/ManageCorsUsing.html>
* Update Bucket Region and Bucket Name in .env file, Access Key and Secret Access Key of IAM User

### Cloudfront Setup

* If you’re creating another CDN service to connect to the bucket, go to cloudfront in AWS and then create a cloudfront.
  + Set the origin of cloudfront to your bucket.
  + The bucket policy has to be updated for CDN to access it. This code will be given at the end of the setup.
  + Set compress objects automatically - yes
  + Viewer protocol - redirect http to https
  + Allowed http method - GET, HEAD
  + Cache key and origin requests
    - Cache policy - Caching Optimized
    - Origin request policy - CORSOriginS3
  + Response Headers policy - Create a policy and set all to allow.
* Once all is done hit create.
* Update S3 Bucket policy
* If any doubts occur use this video for reference - [Reference](https://www.youtube.com/watch?v=kbI7kRWAU-w&ab_channel=SamMeech-Ward)
* Note that for any updates to old images done you have to visit the invalidations page of the cloudfront and invalidate the particular image. This shall refresh the image in the cloudfront service.

### .env Remaining

DB\_SITE = <http://localhost>:<PORT>/images/ (if running on local host else replace localhost:port with the backend website)

NODE\_ENV = development

SITE\_PASSWD = <Password> (Set this based on requirement) (This has to be used to run changes via backend interface)

Once everything is set up and configured in the terminal at the project folder location run “npm run start” or “npm run dev” to make the backend active.

If any “Not allowed by CORS” pops up on the screen then make sure to change the allowedOrigins to the localhost / site that you’re currently running on.

### Backend Forms Exceptions

*dataFill* - Make sure to keep your building name right cause that will be the name that will show up in the transport arrows. Note: The building name is used to recognise the place so make sure that wherever you use it you name it exactly right including nearByBuilding columns.

*deleteData* - Make sure to enter the correct location name in the form to delete all data about it.

*getData* - no input necessary can be used for checking if data was uploaded after using dataFill form.

*patchImage*- Use the right building name and then make sure to upload the correct image.For uploading an image to the description part of a building just use the patchImage form and add “\_1” to the end of your building name and it will add the image.

Same applies for audio to the description part - use patchImage form and add “\_audio.mp3” to the building name and upload mp3 file to the file page.

*patchData* - place data in only inputs you have to make changes in. eg: If you’re increasing the number of buildings nearby you have to increase the previous nearCount by 1 and add the building in the right input column. You can also check the details of your building before changing its data by going to <backendWebsiteURL>/images/name/<building name>. You can install a json formatter extension in chrome to make it more readable.