**Technical Design Document**

ViroTour Server Processing

University of Maryland Global Campus

Software Engineering 670

Spring Cohort 2023

Team B

Logo

Description automatically generated

**Document Control**

**Document History**

|  |  |  |
| --- | --- | --- |
| **Version** | **Issue Date** | **Changes** |
| 0.1 | 2/12/2023 | Initial Draft. |

**Table of Contents**

[1 Introduction 4](#_Toc127134210)

[1.1 Purpose 4](#_Toc127134211)

[1.2 Scope 4](#_Toc127134212)

[1.3 Overview 4](#_Toc127134213)

[1.4 Project Documentation 5](#_Toc127134214)

[1.4.1 Project Suite of Documents 5](#_Toc127134215)

[1.4.2 Document References 5](#_Toc127134216)

[2 System Overview 6](#_Toc127134217)

[3 System Architecture 6](#_Toc127134218)

[3.1 Architectural Design 6](#_Toc127134219)

[3.2 Decomposition Description 7](#_Toc127134220)

[3.2.1 Image Stitching Service Decomposition 7](#_Toc127134221)

[3.2.2 Extract Text Service Decomposition 8](#_Toc127134222)

[3.2.3 Data Storage Decomposition 8](#_Toc127134223)

[3.3 Exception Handling 9](#_Toc127134224)

[3.4 Design Rationale 11](#_Toc127134225)

[4 Data Design 12](#_Toc127134226)

[4.1 Data Description 12](#_Toc127134227)

[4.2 Data Dictionary 14](#_Toc127134228)

[5 Component Design 15](#_Toc127134229)

[5.1 Core Service 16](#_Toc127134230)

[5.2 Editing Service 18](#_Toc127134231)

[5.3 Search Service 20](#_Toc127134232)

[6 Human Interface Design 20](#_Toc127134233)

[7 Requirements Matrix 21](#_Toc127134234)

[Appendix A – Acronyms and Definitions 22](#_Toc127134235)

[Appendix B – Analysis of Alternatives 23](#_Toc127134236)

# Introduction

## Purpose

The purpose of this Technical Design Document (TDD) is to provide insight into the architecture and design of ViroTour. The goal for this document is to convey an overall approach of the system and be able to justify the design choices to our stakeholders.

## Scope

The scope for the ViroTour project for Team B encompasses the server-side development. This TDD will include the system overview, architecture, data design, and requirements for implementation of the virtual tour application.

**In Scope** for Team B:

* Database to store images, locations, hotspots, and text.
* Core Module: Image processing to enable tour navigation.
* Editing Module: Perform modifications on images, locations and hotspots.
* Search Module: Perform text-based search.
* Backend support for any number of Virtual Tours.

For full scope details, please refer to the Project Plan and Software Requirements Specification (SRS).

**Out of Scope** for Team B:

* + Team A’s Responsibilities: User Interface (UI) fundamental design.
* Features related to security, logging, auditing, or monitoring.
* Multiple user profile management.
* User authorization.

## Overview

This TDD consists of the following sections:

* + **System Overview –** A bird’s eye view of the ViroTour Backend.
  + **System Architecture** **–** A detailed design of the components and their relationships.
  + **Data Design** **–** A description of ViroTour’s data layer.
  + **Components Design –** A description of each module’s functional behavior.
  + **Human Interface Design** **–** Contains UI design mockups.
  + **Requirements Matrix –** Represents all functional requirements from SRS and how the components satisfy them.

## Project Documentation

This project documentation section details the documents that will be delivered to the client and the references that the ViroTour Server Processing application utilizes.

### Project Suite of Documents

This is a catalog of ViroTour’s documentation.

| Document | Version | Date |
| --- | --- | --- |
| Project Management Plan (PMP) | 0.2 | 2/11/2023 |
| Software Requirements Specification | 0.2 | 2/11/2023 |
| Technical Design Document | 0.1 | 2/11/2023 |
| Software Test Plan | 0.1 | 2/11/2023 |
| Programmer Guide | TBD | TBD |
| Development and Operations Guide | TBD | TBD |
| User Guide | TBD | TBD |
| Test Report | TBD | TBD |

### Document References

AKX. (2021)*.* Answer to “implement glow filter in CV2 Python.” *Stack Overflow.* [*https://stackoverflow.com/a/68636882*](https://stackoverflow.com/a/68636882)

Matterport. (n.d.). *Matterport Developers.* <https://matterport.github.io/showcase-sdk/index.html>

Matterport. (n.d.). *Graphql schema documentation.* <https://api.matterport.com/model/docs/reference>

OpenCV. (n.d.). *Epipolar geometry*. <https://docs.opencv.org/4.x/da/de9/tutorial_py_epipolar_geometry.html>

Thi, T. (2016). Automated virtual tour. *Oyster.com Tech Blog*. <http://tech.oyster.com/computer-vision-part-2-automated-virtual-tour/>

University of Maryland Global Campus. (n.d.). *Previous projects.* <https://umgc-cappms.azurewebsites.net/previousprojects>

# System Overview

Through the various subsequent sections, we will describe the composition of ViroTour and how it addresses Team B’s various use cases. To summarize, we have an API backed by a strong data storage solution. The technology stack consists of:

* Presentation: The UI implemented in Dart and Flutter.
* Application/Domain: Communication protocol between the front and backend. We are considering using:
  + The Python programming language to develop the application logic.
  + A Python [Flask Server](https://flask.palletsprojects.com/en/2.2.x/) to create a controlled HTTP access-point to our code.
* Database: Stores the data for ViroTour. This will be a SQL and/or Graph-based database.

# System Architecture

## Architectural Design

This architectural design will describe the backend components of the ViroTour Server Processing application. There are four layers of the architecture: Presentation, Application, Domain, and Data Access.

Timeline

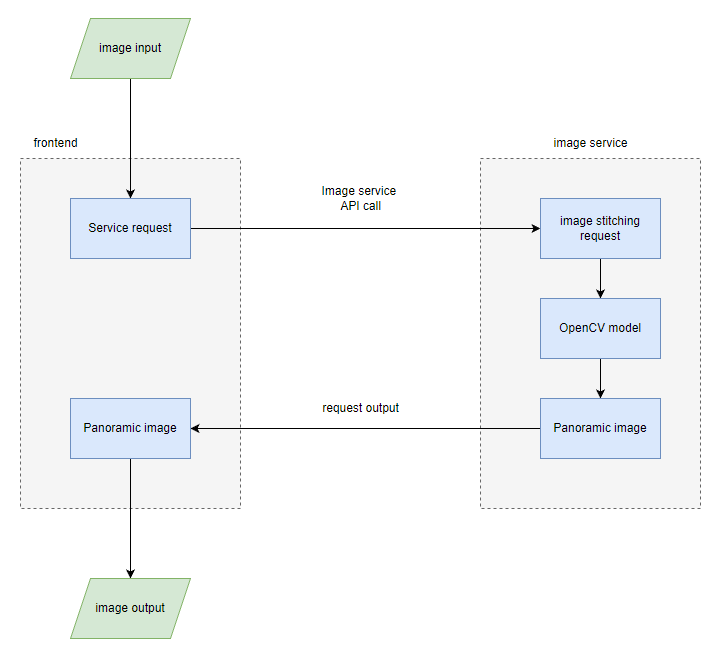
Description automatically generated with medium confidence

The Presentation Layer is the UI. The Application Layer redirects requests to the domain layer. The Domain Layer handles the logic of the application. The Data Layer stores data including hotspot location, extracted text, and filtered images.

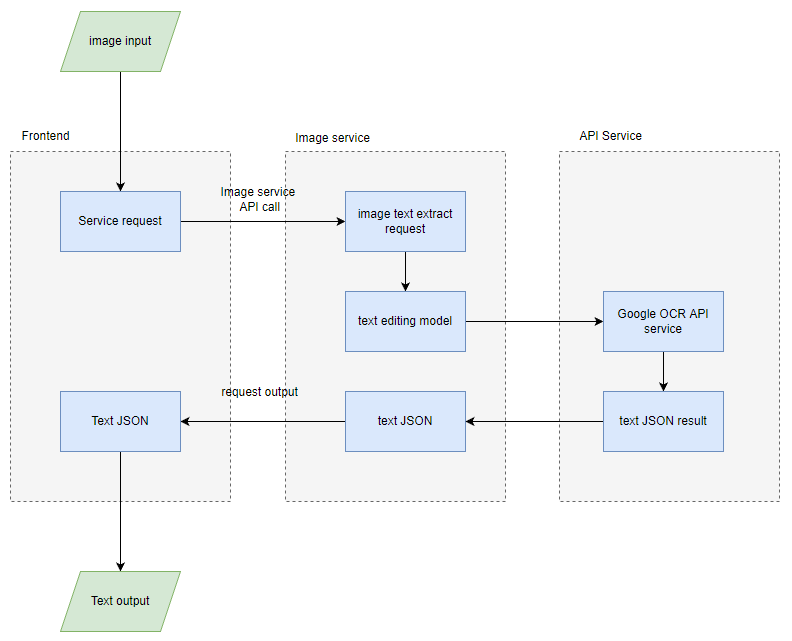
## Decomposition Description

The following 3 diagrams describe the flow of information for the core processing, editing processing, and search processing services.

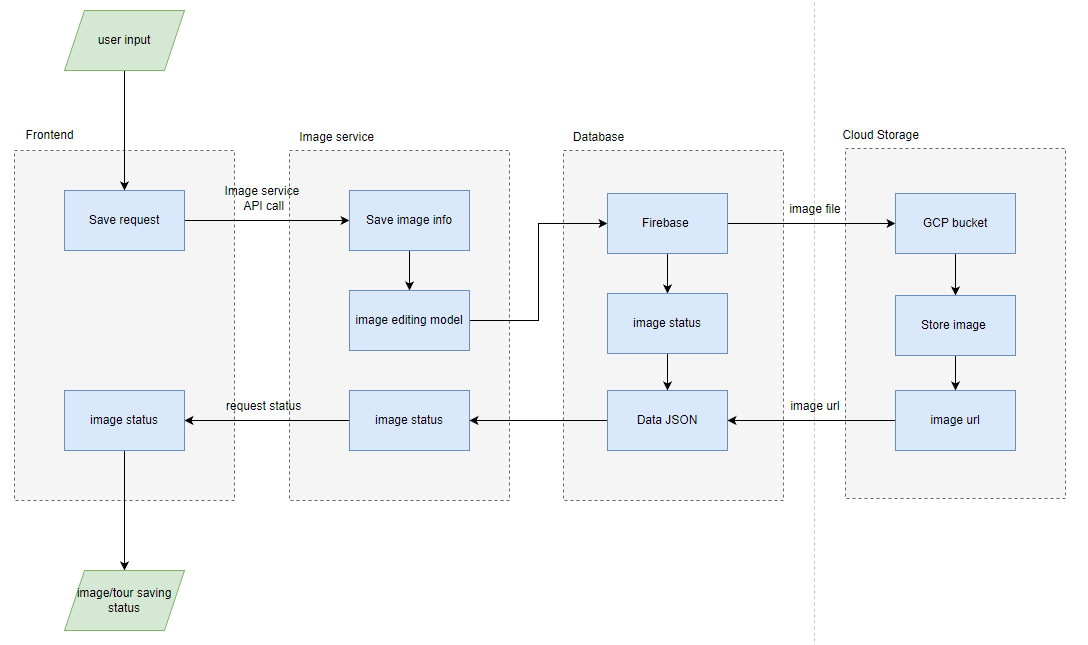
### Image Stitching Service Decomposition



### Extract Text Service Decomposition



### Data Storage Decomposition



## Exception Handling

Exception handling is a way of dealing with the potential errors in our code so our application can gracefully recover from them. The main objective of the exception is to handle the run-time error and prevent the program from terminating abruptly or freezing. Every exception in Dart is a subtype of the pre-defined class Exception.

The try block is utilized to hold the block of code that might be thrown as an exception. The one block is used to when we require specifying the exceptions, and the catch block is used when the handler needs the exception object.

The syntax of typical exception handling in Dart looks like this:

try {  
 //Logical code block  
 } catch (e) {  
 // On exception catch the code here.  
 }

Errors happen all the time in software systems, and these errors can be caused by simple invalid user input, an external system that is not responding, or a programming error. In all these situations, the errors occur at runtime and the application needs to handle them. Otherwise, it crashes and cannot process further requests. Dart provides a powerful mechanism which allows you to handle the exceptional event where it occurred or in one of the higher methods in the call stack.

Errors are events that our program should not normally witness but cannot be entirely prevented from happening. In the exception handling, our main goal is to keep the program safe from errors and the ability to tolerate them without causing disruptions in different parts of the program. In fact, in this section, efforts should be made to make the program resistant to both system errors and errors that the user may make. We know that accidents cannot be prevented, but we can deal with them. There are different ways to deal with this. By writing the right tests and applying them to the software parts, the number of errors during development should be reduced to a minimum and for the occurrence of errors during production, special methodologies should be used. In this project, a central error management system (try catch) can be used in the service layers of the software, and then the cause and time of the error along with the user ID can be stored in the database using an exception translator service. Also, if needed, we can use the Aspect-Oriented Programming methodology and design the interceptor related to the error by implementing the related proxy structure to implement more professional and stronger error detection.

Below is a log of the exceptions managed by ViroTour:

|  |  |
| --- | --- |
| **Exception Name** | **Description** |
| ImageNotFoundException | When accessing a location of the virtual tour, the associated panoramic image is lost or corrupt. |
| UnableToGeneratePanoramaException | This occurs when the image stitching algorithm fails. |
| FileUploadException | This occurs when the ImageUtility fails to upload the image to the file location. |
| UnableToApplyBlurException | This occurs when the image blur algorithm fails. |
| UnableToApplyGlowException | This occurs when the image glow algorithm fails. |
| UnableToComputeTransitionalHotspotException | This occurs when the transitional hotspot computing algorithm fails. |
| UnableToExtractTextException | This occurs when the image text extract fails. |
| UnableToAddHotspotException | This occurs when creating new hotspot fails. |
| UnableToDeleteHotSpotException | This occurs when deleting hotspot fails. |
| UnableToDeleteTourException | This occurs when deleting tour fails. |
| UnableToAddTourException | This occurs when creating new tour fails. |
| InvalidSearchException | When the input search text could not be processed for any other reasons than the system failure. |
| UnableToPerformSearchException | When search cannot be done due to a system issue such as a connection failure. |
| InvalidImageSizeException | It is not suitable for the uploaded image size. |
| InsufficientImageCountException | Occurs when the number of images is low. |
| UnableToUploadException | For when uploading of images has been not successfully performed on the server. |
| UnableToConnectException | When the connection with the server is not established. |
| RuntimeException | When the hotspot is not recognized by the program. |
| UnableToPerformTransactionException | When the program information is not stored in the database. |

*Table 3.2.* Log of ViroTour exceptions.

## Design Rationale

Our design embraces open-source and publicly available software. We use a cross-platform technology called Flutter and Dart to streamline mobile and web development efforts. We are hoping to use custom algorithms and open-source libraries such as OpenCV to implement our processing needs.

We would like to avoid the need to use proprietary software to fulfill the user’s requirements. However, there is a chance we may need to consider using such alternatives. Appendix B – Analysis of Alternatives provides details about every option we have considered so far.

There is still a list of issues we must determine. We hope to use a similar approach to overcome some of our current obstacles:

* + The user may not be satisfied with the result of the panoramas produced by the program. In this case, there may need to be a module for the user to be able to replace those bad images with better ones.
  + If the process of finding hotspots is not done successfully and the algorithm makes a mistake in finding hotspots, then the program will need to report this to the user.

# Data Design

## 4.1 Data Description

The data for the ViroTour application will be stored in a managed location. The data model consists of a list of locations, their geo-spatial coordinates, their neighbors, and other metadata about the location. The figure below displays the different fields.

For each of the requirements, this is how the data will transform:

* When the user uploads images to the server, it stores the location of the images in “images” à “original.”
* When the server computes the panoramic view, it stores the location of the image in “images” à “panoramic.” The “images” à “state” à “settings” is simultaneously set to “panoramic.”
* When the server computes the transition hotspots, it stores the ids of the neighbors in “transitional\_hotspots.”
* When the server extract text from each image, it stores the content in “text” à “content.” The “images” à “state” à “position” is simultaneously set to panoramic.
* When the user adds, edits, or deletes informational hotspots, the server updates “informational\_hotspots” à “position” and “content.”

{

"data": {

"model": {

"locations": [

{

"id": 1,

"position": { "x": 0, "y": 0, "z": 0 },

"images": [

{

"state" : {

"setting" : "blurred",

"filters" : {

"glow" : {

"strength" : 0-255

}

}

},

"original": { "source": [ "img\_lower\_left.png", "img\_lower\_middle.png", "img\_lower\_right.png", "img\_upper\_left.png", "img\_upper\_middle.png", "img\_upper\_right.png" ] },

"panoramic": { "source": [ "img\_panoramic.png" ] },

"blurred": { "source": [ "img\_panoramic\_blurred.png" ] }

}

],

"neighbors" : [

{ "location\_id": 2, "x": 1409, "y": 512 },

{ "location\_id": 3, "x": 609, "y": 652 },

],

"informational\_hotspots" : [

{

"position": { "x": 0, "y": 0, "z": 0 },

"content": ["<html><body> … </body></html>"]

}

],

"text" : [

{

"position": { "x": 0, "y": 0, "z": 0 },

"content": ["This painting depicts the struggles of…"]

}

]

}

]

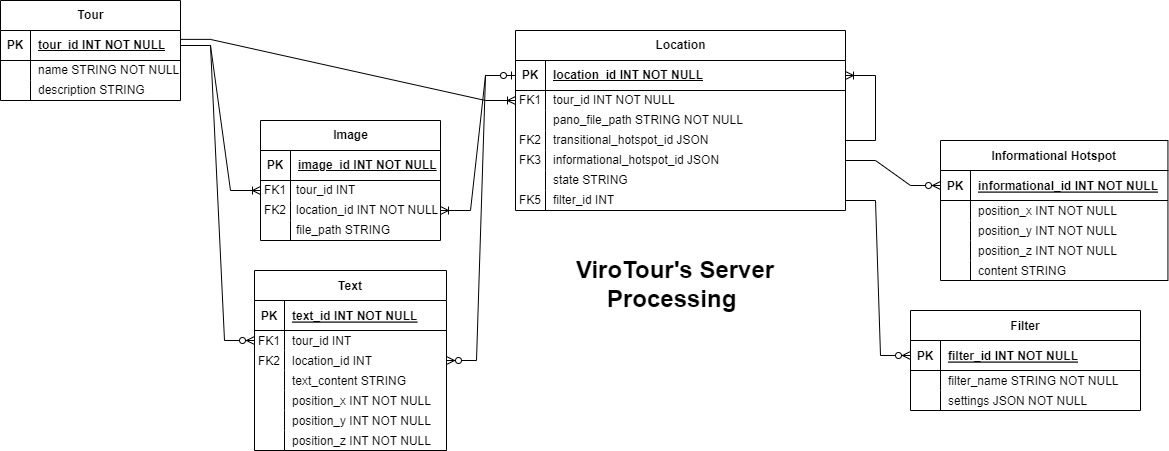
}

}

}

*Figure 4.1*. Data layout.

The Entity-Relationship Diagram below reflects the rationale of the data layout:



*Figure 4.2.* Entity relationship diagram.

## 4.2 Data Dictionary

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Type** | **Description** |
| blurred | Object | Contains information about the blurred image |
| content | Array of strings | Each string represents the content of the informational hotspot |
| data | Object | Contains the model information |
| exception | Object | Information about exception(s) that happened in app (including date, issue, level) |
| filters | Object | Contains information about the applied filters |
| glow | Object | Contains information about the glow filter |
| id | String | Unique identifier of the location |
| images | Array of objects | Image associated with the location |
| informational\_hotspots | Array of objects | Each object represents an informational hotspot associated with the location |
| locations | Array of objects | An array of objects, each representing a location |
| log | Object | Information about user command (including date, command, user info, level) |
| model | Object | Contains information about the locations |
| original | Object | Contains information about the original image |
| panoramic | Object | Contains information about the panoramic image |
| position | Object | Represents the 3D position of the location |
| radius | Number | Representation of the radius of the glow filter |
| setting | String | Setting of the image (either "blurred" or "clear") |
| source | Array of strings | Represents the source of an original image |
| state | Object | State of the image (contains the “setting” and “filters” properties and respective values) |
| strength | Number | Representation of the strength of the glow filter |
| text | Array of objects | Each object represents a text object associated with the location |
| transitional\_hotspots | Array of strings | Each string represents an ID of a location to which the current location can transition to |
| x | Number | Position along the x-axis |
| y | Number | Position along the y-axis |
| z | Number | Position along the z-axis |

# Component Design

## Core Service

Class Name: ImageUtility

Class Description: Provides the ability to upload images to a file location and stitch all images located within the file location to generate a panorama.

Class Attributes:

* + Public imageID: string
  + Pubic imagePath: string
  + Public imageState: string
  + Public locationID: string

Class Modifier:

* + Public

Class Constructors:

* + ImageUtility(): No parameter required

Class Methods:

* + imageUpload(string imagePath): Uploads all images to specific file location and sets initial state of images to “original.”
  + createLocationID(string imagePath): Generates a unique location id based on file location.
  + getImage(string imageID): Retrieve the image based on image id.
  + stitchImages(string imagePath): Stitches all images from a specific file location using OpenCV algorithm.
  + savePanoramicImage(string imagePath): Saves the newly generated panoramic image at the specific file location and sets state of the image to “panorama.”
  + getLocationID(string locationID): Retrieve the location id that the image or panorama is associated to.

User-defined Exceptions:

* + UnableToGeneratePanoramaException: Occurs when the image stitching algorithm fails.
  + FileUploadException: Occurs when the ImageUtility fails to upload the image to the file location.

Class Name: ImageProcessor

Class Description: Provides the ability to apply filters on a panorama.

Class Attributes:

* + Public imageID: string

Class Modifier:

* + Public

Class Constructors:

* + ImageProcessor(): No parameter required

Class Methods:

* + setBlur(string imageID): Applies the blur filter on a panoramic image with human faces by executing an OpenCV algorithm and sets the state of the panoramic image to “blur.”
  + setGlow(string imageID): Applies the glow filter on a panoramic image.
  + getImage(string imageID): Retrieve the image based on image id.
  + savePanorama(string imagePath): Saves the newly generated panoramic image at the specific file location.

User-defined Exceptions:

* + UnableToApplyBlurException: Occurs when the image blur algorithm fails.
  + UnableToApplyGlowException: Occurs when the image glow algorithm fails.

Class Name: LocationProcessor

Class Description: Provides the ability to retrieve panoramic image location data and find transitional hotspots.

Class Attributes:

* + Public locationID: string
  + Public x\_coordinate: string
  + Public y\_coordinate: string
  + Public z\_coordinate: string
  + Public hotSpotType: string

Class Modifier:

* + Public

Class Constructors:

* + LocationProcessor(): No parameter required

Class Methods:

* + getLocationID(string x, string y, string z): Retrieve the location id based on coordinates.
  + getHotspots(string locationID, string hotspotType): Retrieves a collection of transitional or informational hotspots based on locationID and type of hotspot (informational or transitional).
  + computeTransitionalHotspot(string locationID): Generates transitional hotspots based on images associated with a specific location id by executing an OpenCV algorithm.
  + saveTransitionalHotspot(string locationID): Stores and associates a transitional hotspot to a specific location id.

User-defined Exceptions:

* + UnableToComputeTransitionalHotspotException: Occurs when the transitional hotspot computing algorithm fails.

Class Diagram:



## Editing Service

Class Name: TextProcessor

Class Description: Provides the ability to extract text from input image.

Class Attributes:

* + Public x\_coordinate: float
  + Public y\_coordinate: float
  + Public textID: string

Class Modifier:

* + Public

Class Constructors:

* + TextProcessor(); No parameter required

Class Methods:

* + extractText(string imagePath): Extract the text and corresponding coordinates from the image.

User-defined Exceptions:

* + UnableToExtractTextException(): Occurs when the image text extract fails.

Class Name: TourProcessor

Class Description: Provides the ability to edit tour details.

Class Attributes:

* + Public tourID: string
  + Public hotspotID: string
  + Public x\_coordinate: float
  + Public y\_coordinate: float

Class Modifier:

* + Public

Class Constructors:

* + TourProcessor(); No parameter required

Class Methods:

* + addHotSpot();
  + EditHotSpot(String hotspotID, float x, float y);
  + DeleteHotSpot(String hotspotID);
  + DeleteTour(String TourID);
  + AddTour(String TourID);

User-defined Exceptions:

* + UnabletoAddHotspotException(): Occurs when creating new hotspot fails.
  + UnabletoDeleteHotSpotException(): Occurs when deleting hotspot fails.
  + UnabletoDeleteTourException(): Occurs when deleting tour fails.
  + UnabletoAddTourException(): Occurs when creating new tour fails.

## Search Service

Class Name: PanoramicSearchService

Class Description: Provide the user the ability to search for any panoramic images by taking in a textual query.

Class Attributes:

* + Private imagesCollection: List of panoramic image objects

Class Modifier:

* + public

Class Constructors

* + PanoramicSearchService(): No parameter required

Class Methods

* getImages()**:** Retrieve all the available image objects.
* searchImages(searchText: string**):** Search for image objects based on the user search input text, and assign them to the collection.

User-defined Exceptions:

* InvalidSearchException**:** When the input search text could not be processed for any other reasons than the system failure.
* UnableToPerformSearchException**:** When search cannot be done due to a system issue such as a connection failure.

Class diagram:



# Human Interface Design

The human interface design of the ViroTour Server Processing application will be handled by Team A as Team A handles the presentation layer of the application that the user will be interacting with. Team B and this technical design document do not have any direct interaction with the user.

# Requirements Matrix

The requirements matrix table below identifies the software requirements defined in the ViroTour SRS. These requirements are satisfied by the specified component from the Component Design Section in this TDD.

|  |  |  |  |
| --- | --- | --- | --- |
| **SRS – Requirement Number** | **Requirements Description** | **Component Number** | **Component Name** |
| 3.1.1.1 | The user uploads an image to the server. | 5.1 | Core Service |
| 3.1.1.2 | Panoramic images are created. | 5.1 | Core Service |
| 3.1.1.3 | Determine transition hotspots. | 5.1 | Core Service |
| 3.1.2.1 | Extract text from images. | 5.2 | Editing Service |
| 3.1.2.2 | Detect and blur human faces from images. | 5.2 | Editing Service |
| 3.1.2.3 | Apply filters. | 5.2 | Editing Service |
| 3.1.2.4 | Add hotspot. | 5.2 | Editing Service |
| 3.1.2.5 | Edit hotspot. | 5.2 | Editing Service |
| 3.1.2.6 | Delete hotspot | 5.2 | Editing Service |
| 3.1.3.1 | Ability to search for extracted text from requirement #3.1.2.1. | 5.3 | Search Service |

Appendix A – Acronyms and Definitions

Common terminology is seen and presented throughout this document. To remove any ambiguity, below is a list of terms and their intended meanings:

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Authorization | It is a term in the field of security, and it deals with the control of the permission of the user's access level. |
| Consistent | The compatibility of the designed software with the business use cases in the project. |
| Cross-functional | Functional requirements that the user is not directly related to but are necessary for the correct operation of the software. |
| Dart | An object-oriented programming language developed by Google. |
| Digital signature | It is a method of verifying a message that includes public and private keys that are unique to each person in asymmetric encryption. |
| Flutter | An open-source framework developed by Google. |
| Hotspot | A preset location in the app that acts as a destination point of interest users can interact with. |
| Image stitching | Process of combining multiple pictures that overlap and producing a panorama image. |
| Image processing | The process of converting an image into a digital form and performing specific operations on it. |
| Maintaining | All processes are done after product delivery to ensure performance and improve efficiency and reduce product errors. |
| Panorama | An unobstructed view of an area that is visible in every direction. |
| Scrum | It is an incremental methodology for managing software projects and is considered an agile methodology. |
| Software as service | A method for providing software that enables access to software data from any device with an Internet connection and a web browser. |
| Service | Delivery of a job (developed by a program department) to achieve a specific goal. |
| TDD | Technical Design Document. |
| UI | User Interface. |
| UX | User Experience. |
| Upload | The process of transferring files from the user's local space to the server space. |
| Virtual tours | A series of rotating images that are connected to form a complete 360-degree view of a place or location. |

Appendix B – Analysis of Alternatives

This section will discuss the alternative approaches that were considered, but not selected for further development:

**Option #1: Matterport SDK**

Matterport’s software includes automatic hotspot detection, and a discovery effort was undertaken to examine how to utilize it. By creating a free account and generating an SDK and API key, we can access Matterport’s server to extract data related to the virtual tour.

There are a few different approaches we can take through Matterport. The table below summarizes the technical options for addressing each use case:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Option 1:**  **Matterport Native App + Matterport API** | **Option 2:**  **Matterport API (data collection) + Custom VR Open-source** | **Option 3:**  **Matterport SDK (iframe)** |
| **Complexity** | Low | Hard | Medium |
| **Number of Virtual Tours Supported** | 1 VR Tour | Any number | 1 VR Tour |
| **Use Case Coverage** | 1 Use Case Missing | All | All |
| **Academic?** | Yes | Yes | Somewhat |
| **Upload Images to Server as Groups Which Represent a Location ("image group" or "location")** | Matterport Native App | Matterport Native App | Matterport SDK |
| **Determine and Provide a Panoramic View of a Location.** | Matterport Native App | Matterport API ("panos" -> openCV -> custom storage "panos\_stitched") | Matterport SDK |
| **Determine and Provide Transitions Between Locations ("transition hotspot").** | Matterport Native App | Matterport API ("neighbors" -> open-source app) | Matterport SDK |
| **Detect Text from Images for each Location and Store Results** | Matterport API -> Get Images ("panos") -> Detect Text -> Add Matterport Tags with Coordinates automatically | Matterport API ("panos" -> openCV -> custom storage "text") | Matterport SDK |
| **Blur Human Faces and Skin from All Images** | Not possible to change images | Matterport API ("panos\_stitched" -> openCV -> custom storage "panos\_blurred") | Matterport SDK |
| **Ability to Apply Filters to Images and Store Results** | Matterport API -> Get Images ("panos") -> Detect Text -> Add Matterport Tags with Coordinates automatically | Matterport API ("panos" -> openCV -> custom storage "panos\_filtered") | Matterport SDK |
| **Ability to Add a Hotspot** | Matterport Native App | open-source app | Matterport SDK |
| **Ability to Edit a Hotspot** | Matterport Native App | open-source app | Matterport SDK |
| **Ability to Delete a Hotspot** | Matterport Native App | open-source app | Matterport SDK |
| **Ability to Search All Text from B-2.1** | Matterport Native App | open-source app | Matterport SDK |

Options 1 is not possible to use because it does not fulfill all use cases. Option 3 is not favorable because we must deal with a proprietary library. Option 2 will be used as a backup plan if all else fails.

References:

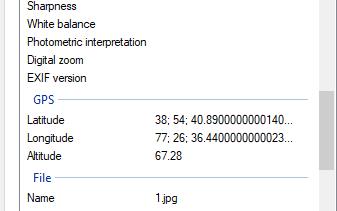
Matterport. (n.d.). *Create a free account*. <https://buy.matterport.com/free-account-register>

Matterport (2023). *How to use Matterport APIs*. <https://support.matterport.com/s/article/How-To-Use-The-Matterport-API?language=en_US>

**Option #2: GPS Coordinates / Longitude and Latitude**

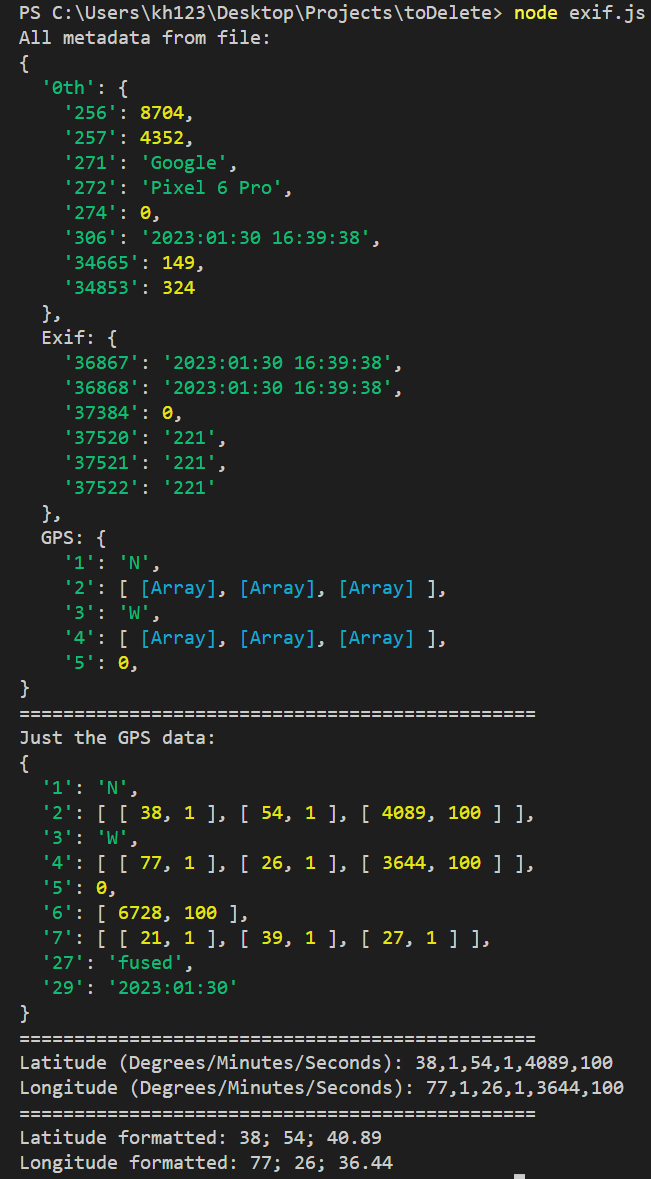
Geographic metadata, specifically longitude and latitude, were considered of interest to implement the transitional hotspots between the panoramic images. The thought process involved converting the extracted coordinates and converting it to a x and y position on the equirectangular projection images. If possible, then from any currently selected hotspot, the locations of the other transitional hotspots would be calculated and be present in current view.

One effort concluded that images taken with a mobile device had enclosed metadata:



*Figure C.2.1.* Jpg image property details.

To extract the latitude and longitude, we found two libraries/packages to do the job: Exif (via Python) and Piexifjs (via Node.js). Implementation via the Piexfjs library to extract latitude and longitude is possible, as seen below:



*Figure C.2.2.* Retrieving latitude and longitude metadata from jpg via Piexjfs.

However, the issue with GPS being unreliable indoors marked this approach not favorable.

References:

DeVilla, J. (2021). JavaScript and photos: read, edit, and erase location and other Exif metadata. *Auth0*. <https://auth0.com/blog/read-edit-exif-metadata-in-photos-with-javascript/>

PyPI – The Python Package Index (2023). *Exif 1.6.0*. <https://pypi.org/project/exif/>

**Option #3: Extraction / Exporting Job Data from Matterport Mobile App**

Matterport has a mobile application that provides functionality to take panoramic pictures and store them in archives. The intent was to see if extraction of the images’ metadata was possible, seeing if it could help in determining a solution to handling the placement of transitional hotspots. However, extracting the data (.mpp files) resulted in bytecode, putting a stop to examining this analysis:

A picture containing text

Description automatically generated

*Figure C.3.1.* Matterport’s extracted job data in bytecode*.*

**Option #4: Drone Mapping Technology**

Utilizing a drone to create the panoramic images at points of interest was shortly looked at, finding an open-source implementation on GitHub that used simultaneous localization and mapping (SLAM). This concept would generate the boundary on an area and with the drone going to selected point of interests determined within the boundary, would generate the panoramic 360° images that would be used for transitional hotspots.

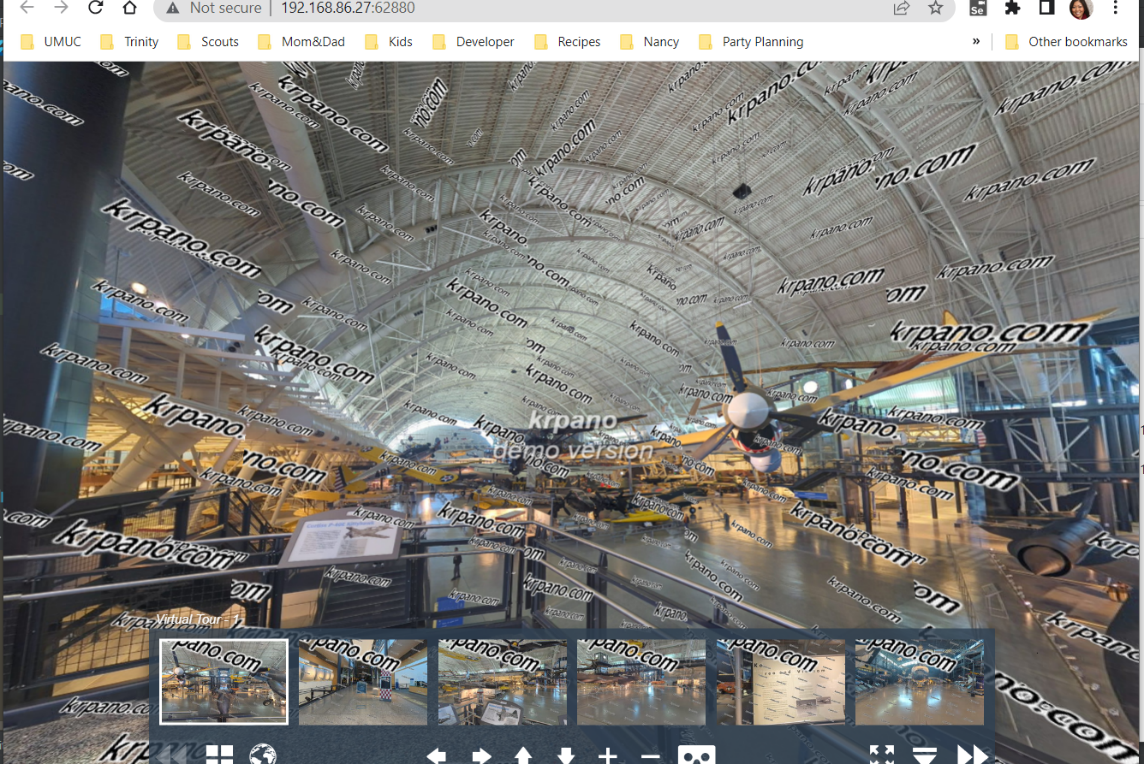
With this solution, a drone would be needed for any images to be added to the application. Also, with phones being a more practical implementation of taking panoramic images, this choice was not found to be feasible.

Reference:

GitHub. (n.d.). *Auto\_Virtual\_Tour\_Gen*. <https://github.com/tau-adl/Auto_Virtual_Tour_Gen>

**Option #5: Krpano Panorama Viewer**

Krpano was a potential software solution to our virtual tour needs, providing required functionality including (but not limited to): transitioning between set hotpots, viewing in all 360 degrees, and zooming in and out of image. Incorporating the software allowed initial interactions, as seen below:



*Figure C.5.1.* Krpano with panoramic pictures taken at National Air and Space Museum.

While functionality was working as intended, the software must be bought to utilize the software without the project watermark presented within our panoramic images. With that, Krpano was not selected for further implementation.

Reference:

Krpano. (n.d.). *Krpano panoramic viewer*. <https://krpano.com/home/>