Boston University
Electrical & Computer Engineering
EC464 Capstone Senior Design Project

User's Manual

MUSEUMMATE

Submitted to

Thomas DC Little
8 Saint Mary's St.
Boston MA 02215
617-353-9877
tdcl@bu.edu

by

Team 7
MuseumMate

Team Members

John Culley jculley@bu.edu
Kwadwo Osafo kosafo@bu.edu
Kai Imery kimery@bu.edu
Ananth Sanjay ananths@bu.edu
Yangyang Zhang yz21@bu.edu

Submitted: 04-15-2024

MuseumMate

Table of Contents

(Right click on Table of Contents to update fields and page numbers automatically)

Executive Summary

MuseumMate is an advanced indoor navigation and exhibit interaction system, designed to enrich museum visits. Upon entering a museum, visitors receive a handheld device, the TourTag, which detects Ultra-wideband (UWB) signals from beacons placed throughout the premises. This device transmits signal strengths to a server to pinpoint the visitor's location, facilitating precise navigation through a user-friendly web application. Additionally, RFID tags on exhibits interact with the TourTags, allowing visitors to automatically access rich multimedia content about the exhibits they are near, thus enhancing their educational and interactive experience in the museum. This system seamlessly integrates efficient navigation with engaging exhibit exploration, revolutionizing the traditional museum visit.

1 Introduction

Museums are pivotal for cultural preservation and are one of the most popular venues for education and entertainment, attracting millions of visitors annually. Despite their importance, visitors often face challenges such as navigation difficulties due to complex layouts and congestion at popular exhibits. To address these issues, MuseumMate introduces a cutting-edge solution designed to enhance the visitor experience through seamless navigation and enriched multimedia engagement.

Context and Technological Foundations

MuseumMate utilizes a blend of Ultra-Wideband (UWB) and Radio Frequency Identification (RFID) technologies to streamline visitor flow and provide immersive educational content. The system's core components include:

- UWB Beacons (ESP32 UWB DW3000): These beacons are utilized with trilateration to provide location accuracy within 10 centimeters , facilitating real-time, room-by-room navigation and helping to manage museum traffic by suggesting alternative routes to avoid congestion.
- RFID Tags: Attached to each exhibit, these tags work in tandem with TourTags—portable user devices that visitors carry. When a visitor approaches an exhibit, the TourTag scans the RFID tag to fetch multimedia content related to the exhibit.
- Server Technology:
    - Node.js with Express.js: Powers the backend infrastructure facilitating efficient data handling and communications.
    - MinIO: A high-performance storage solution that manages the multimedia files which visitors access through their TourTags.
    - InfluxDB: A time-series database optimized for storing and retrieving real-time metrics that help museum staff monitor and optimize visitor traffic patterns.
- TourTag Device: This custom-designed portable device features a compact PCB that includes all necessary hardware ports and connects to visitors' phones via a MagSafe-compatible enclosure, enhancing portability and ease of use.

Special Features from the User's Perspective

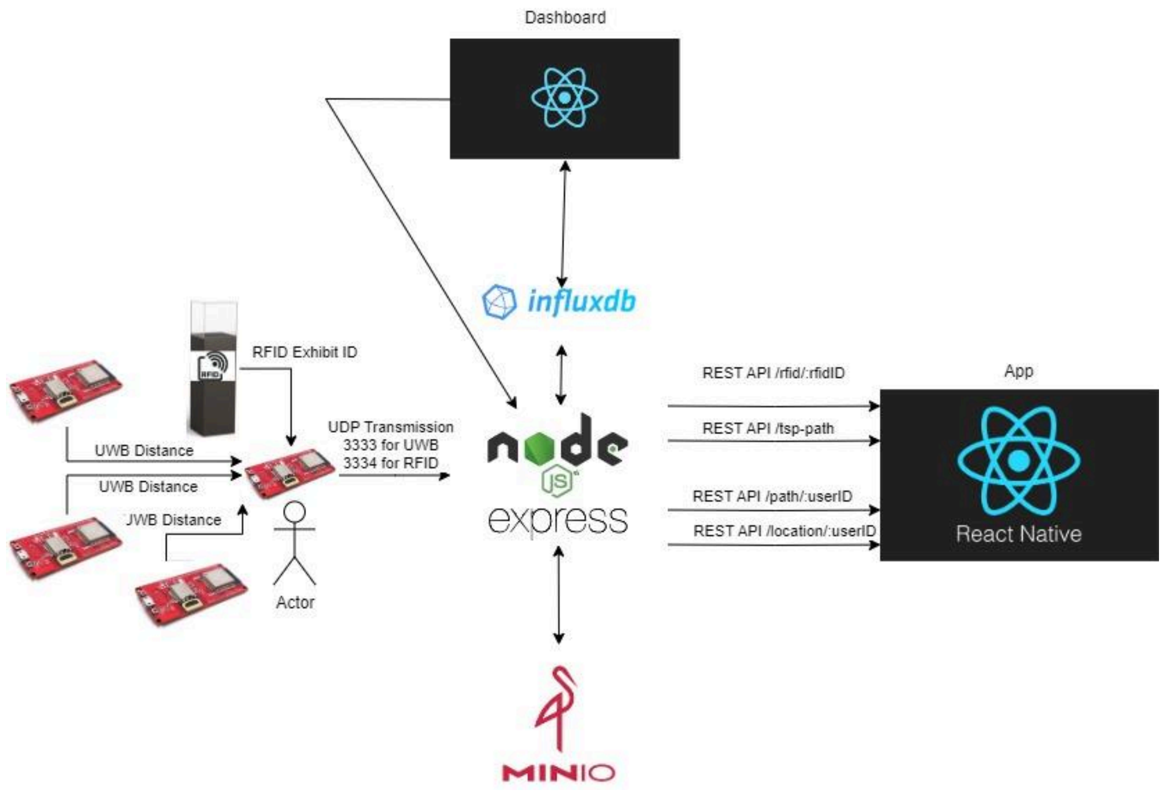
MuseumMate is not just a navigation tool but a comprehensive museum assistant:

- Advanced Navigation Algorithms: Utilizing Dijkstra's and the Traveling Salesman Problem (TSP) algorithms, MuseumMate provides optimized routing solutions that dynamically adjust to the visitor's location and crowd conditions, ensuring minimal wait times and balanced visitor distribution throughout the museum.
- RFID Multimedia Support: Each exhibit's RFID tag allows visitors to access a wide array of multimedia resources, such as audio descriptions, subtitled videos, and high-resolution images, enhancing the learning experience. This feature is particularly beneficial for visitors with physical impairments, as it supports 13 different languages, text-to-speech, and a ChatGPT AI Assistant.

- Real-Time Data Processing: With real-time analytics powered by InfluxDB, MuseumMate delivers instant insights to museum staff, enabling them to make informed decisions about exhibit openings and traffic management.
- Interactive Dashboard: A React-based web application that uses ApexCharts to visualize data from InfluxDB, providing staff with user-friendly interfaces to monitor museum operations effectively.
- Mobile Application: Developed using React Native, the mobile app is an integral part of the visitor experience, offering intuitive navigation, exhibit information, and easy access to personalized multimedia content.

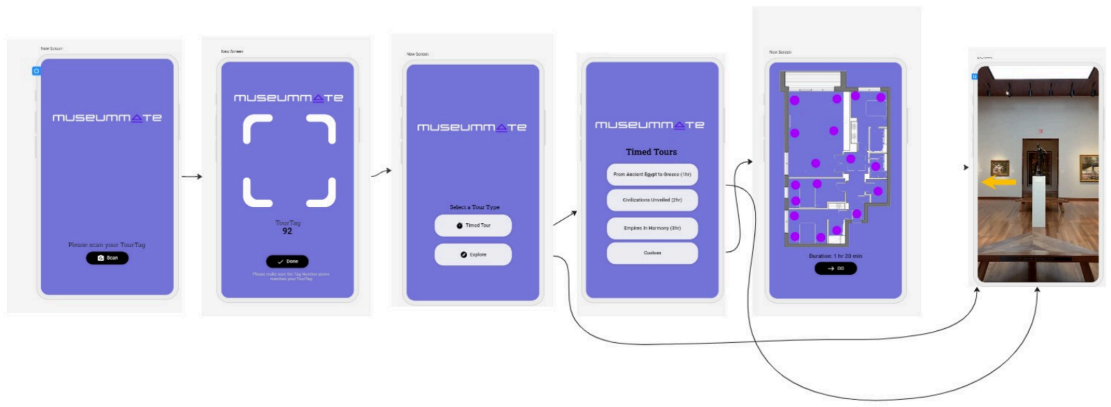
2 System Overview and Installation

2.1 Overview block diagram

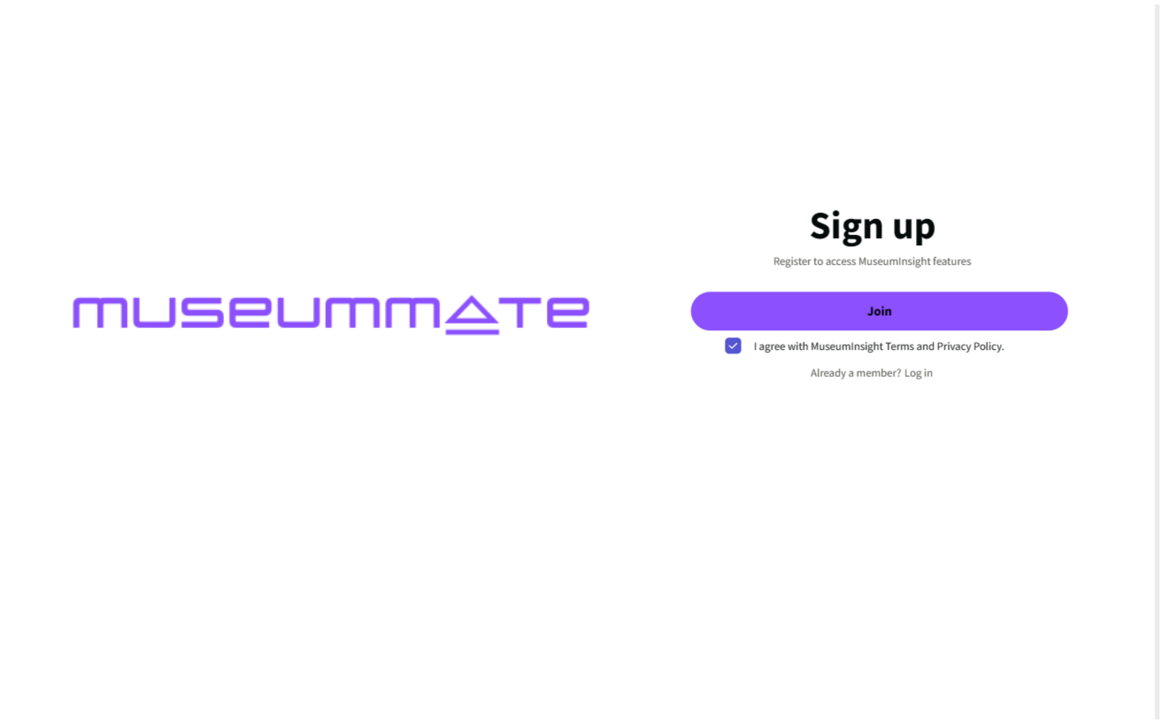


2.2 User interface.

Mobile App



Dashboard

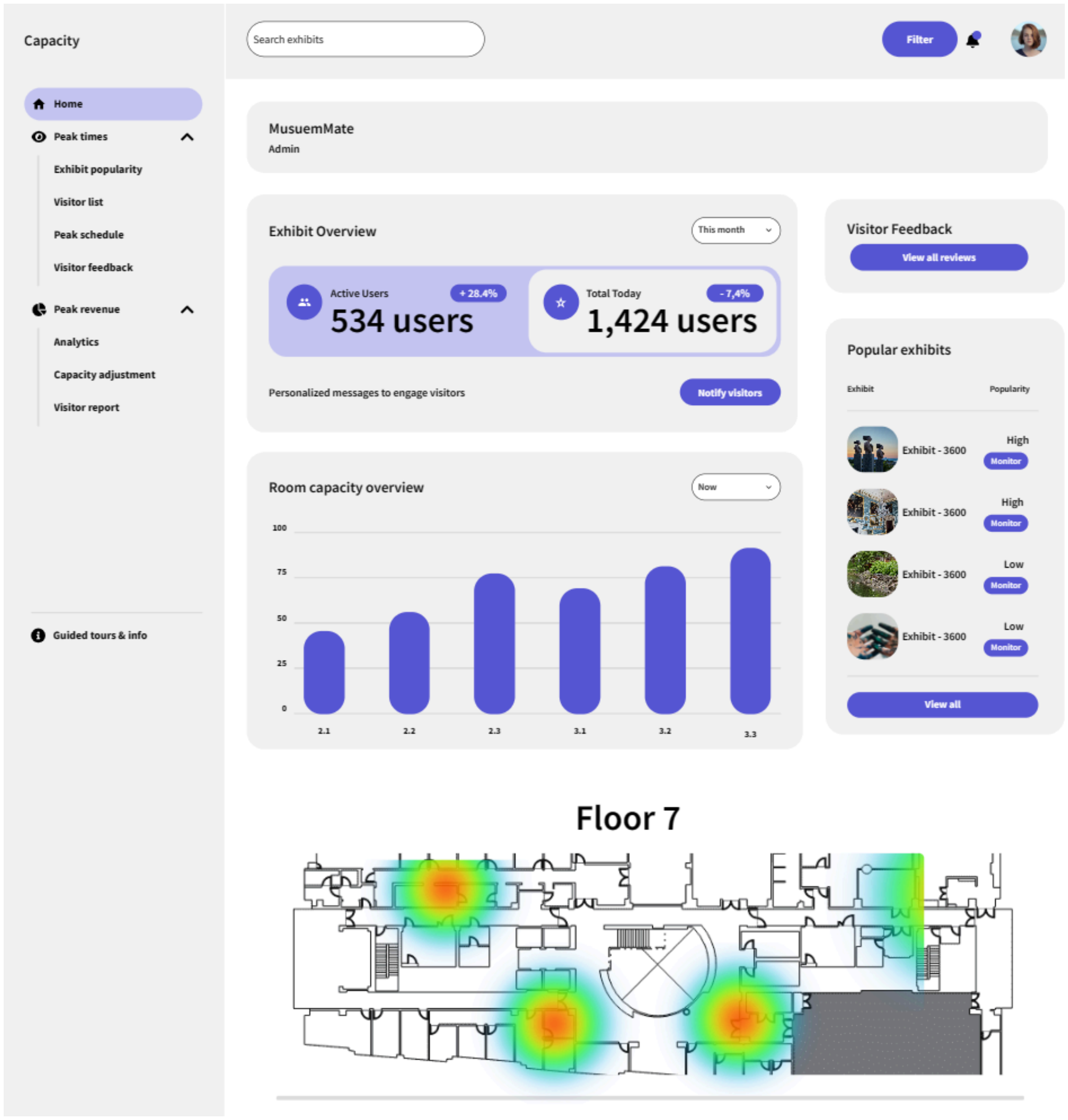


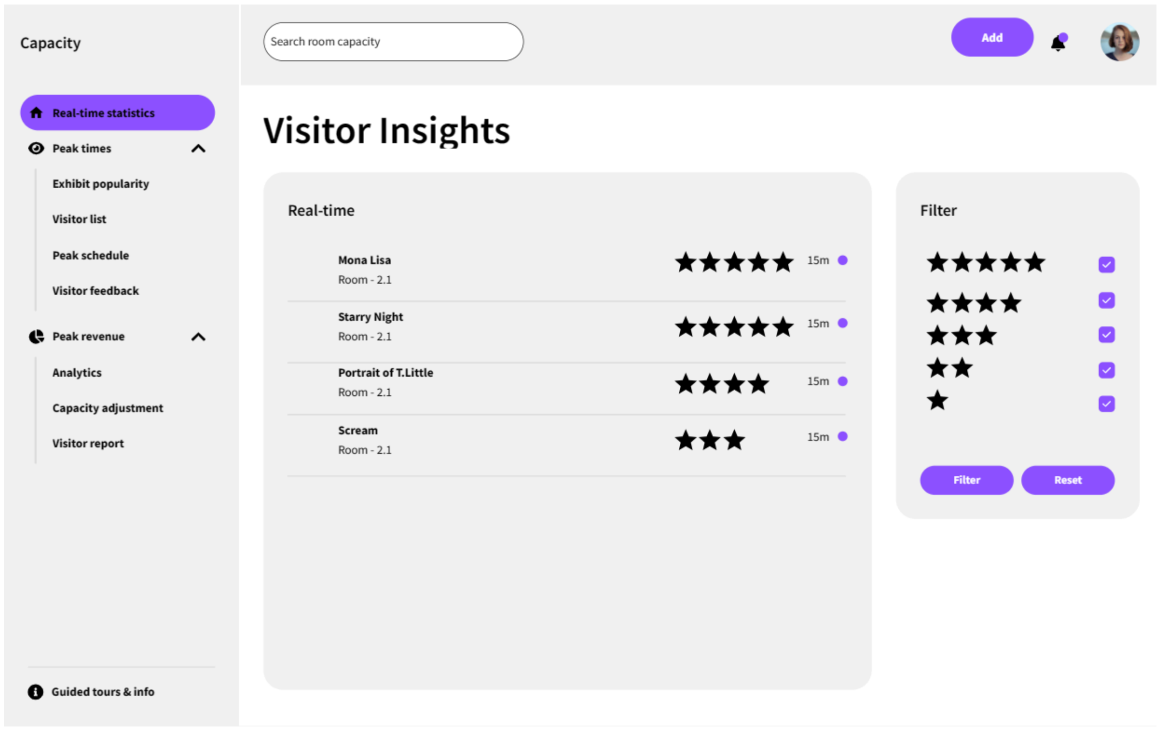
Sign up

Register to access MuseumInsight features

✓ I agree with MuseumInsight Terms and Privacy Policy.

Already a member? Log in


Capacity
Search exhibits
Filter
Home
Peak times
Exhibit popularity
Visitor list
Peak schedule
Visitor feedback
Peak revenue
Analytics
Capacity adjustment
Visitor report
Guided tours & info
MusuemMate
Admin
Exhibit Overview
This month
Active Users
+ 28.4%
534 users
Total Today
- 7,4%
1,424 users
Personalized messages to engage visitors
Notify visitors
Visitor Feedback
View all reviews
Popular exhibits
Exhibit
Popularity
Exhibit - 3600
High
Monitor
Exhibit - 3600
High
Monitor
Exhibit - 3600
Low
Monitor
Exhibit - 3600
Low
Monitor
View all
Room capacity overview
Now
100
75
50
25
0
2.1
2.2
2.3
3.1
3.2
3.3
Floor 7



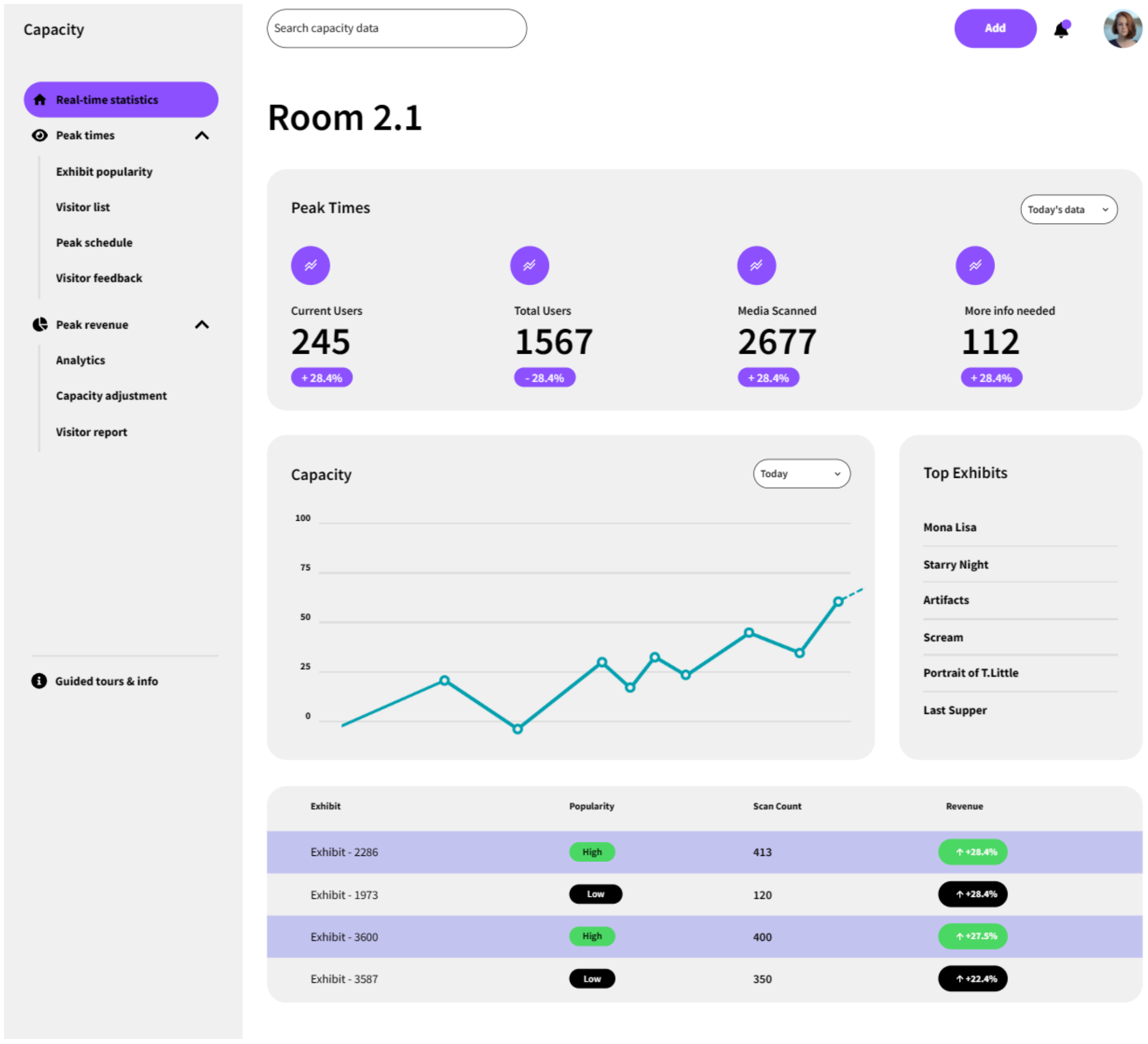
Visitor Insights

Real-time

Mona Lisa Room - 2.1	★★★★★	15m	●
Starry Night Room - 2.1	★★★★★	15m	●
Portrait of T.Little Room - 2.1	★★★★	15m	●
Scream Room - 2.1	★★★	15m	●

Filter

★★★★★	✓
★★★★	✓
★★★	✓
★★	✓
★	✓



Capacity

Search capacity data

Add

- Real-time statistics
- Peak times
  - Exhibit popularity
  - Visitor list
  - Peak schedule
  - Visitor feedback
- Peak revenue
  - Analytics
  - Capacity adjustment
  - Visitor report
- Guided tours & info

Room 2.1

Peak Times — Today's data

Current Users	Total Users	Media Scanned	More info needed
245	1567	2677	112
+ 28.4%	- 28.4%	+ 28.4%	+ 28.4%

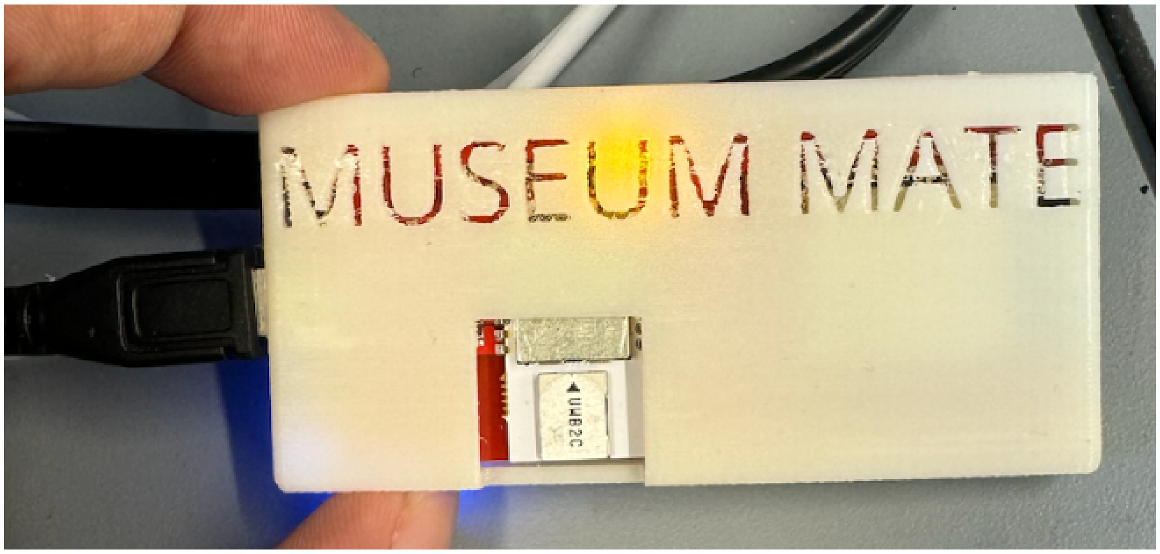
Capacity — Today

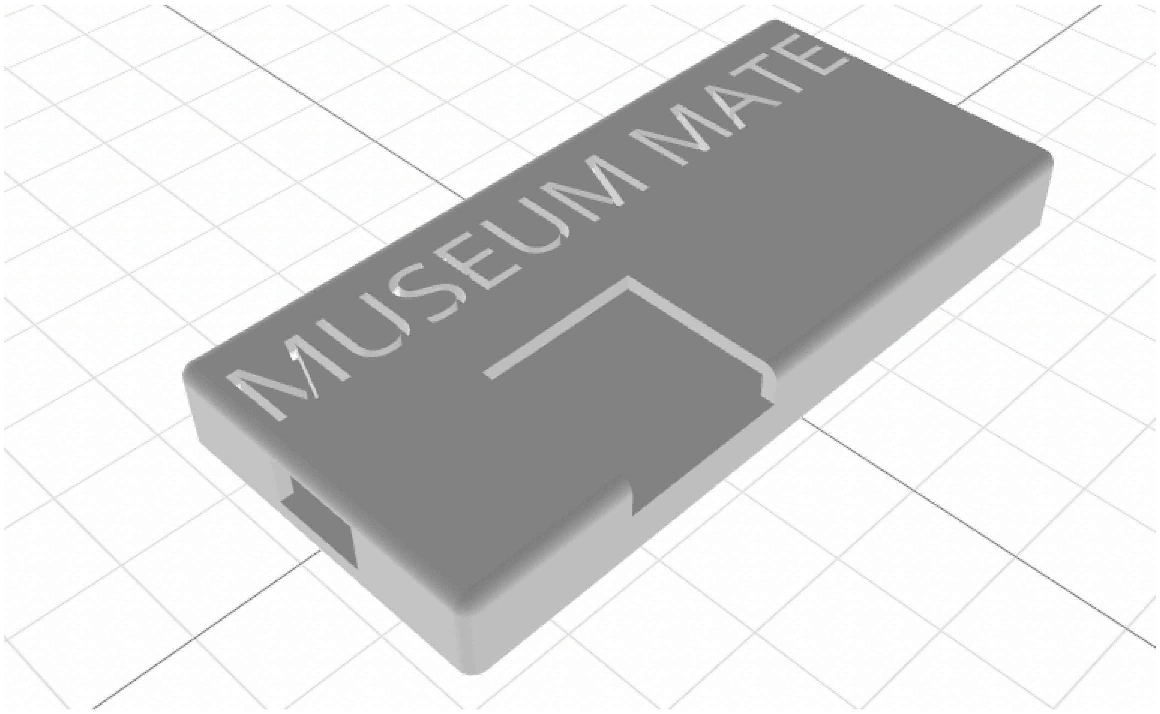
Top Exhibits

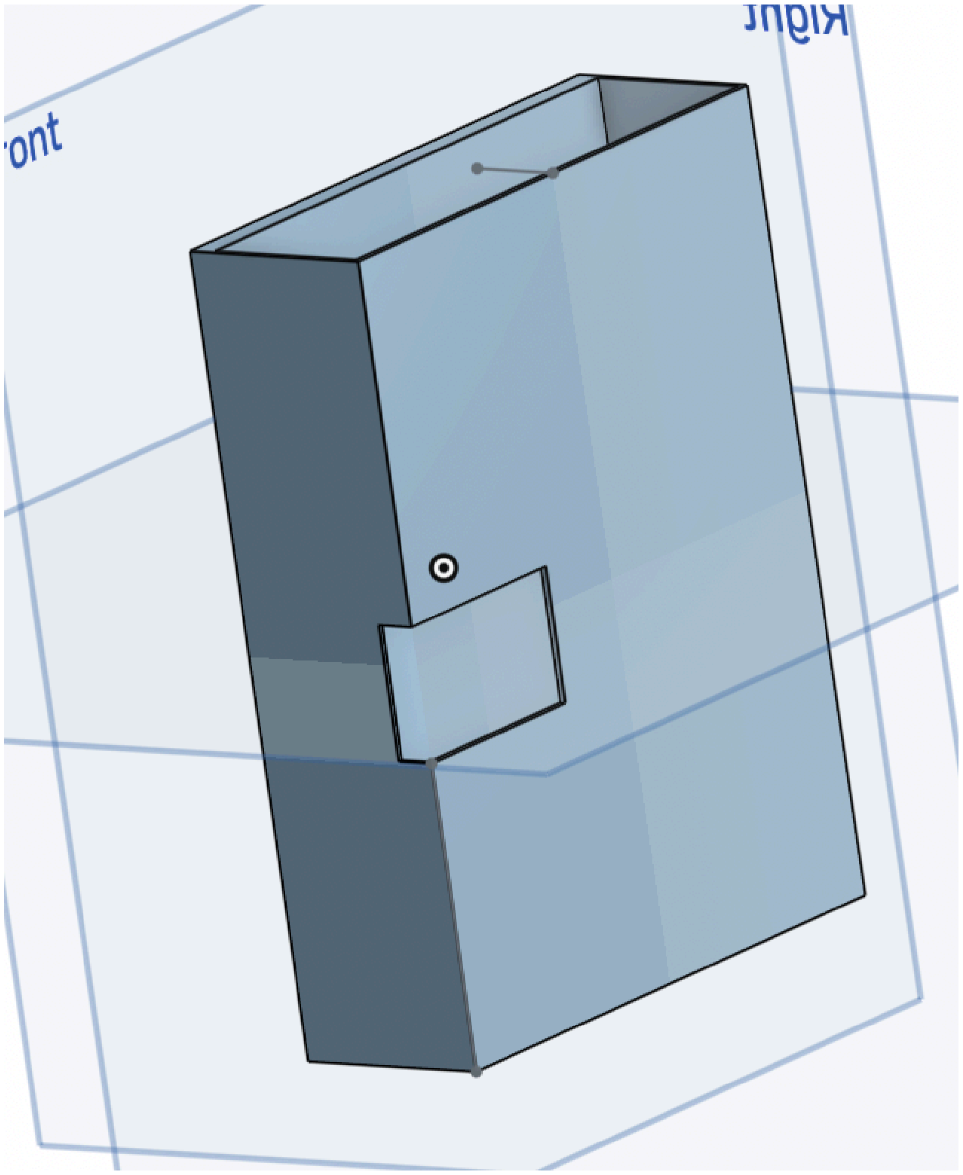
- Mona Lisa
- Starry Night
- Artifacts
- Scream
- Portrait of T.Little
- Last Supper

Exhibit	Popularity	Scan Count	Revenue
Exhibit - 2286	High	413	↑ +28.4%
Exhibit - 1973	Low	120	↑ +28.4%
Exhibit - 3600	High	400	↑ +27.5%
Exhibit - 3587	Low	350	↑ +22.4%

2.3 Physical description.







2.4 *Installation, setup, and support*

2.4.1 Hardware Installation

The MuseumMate system comprises three key hardware components: TourTag, Beacon, and RFID Stickers. To ensure a seamless setup experience, please follow the detailed steps provided for each component below.

2.4.1.1 TourTag Setup

- WiFi Configuration: Begin by configuring the WiFi settings to enable communication with the server. Open the MuseumMateWifi.h file in your preferred code editor and enter the necessary WiFi parameters, including the network SSID and password. Additionally, specify the server's IP address within this file to establish a connection.
- User ID Configuration: The TourTag device requires a unique User ID to operate. Navigate to the common_definition.h file and input the designated User ID. This step is crucial for personalizing the device to your specific needs.

- Programming the Device: With the configurations set, use the Arduino IDE to upload the program to the TourTag device. Ensure a successful transfer by following the IDE's instructions for uploading code to hardware.
- Charging: The TourTag is powered by an internal lithium battery. To charge the device, connect it to a power source using a Mini B USB power adapter. The device automatically begins operation upon disconnection from the charger.
- MagSafe Support: For convenience, the TourTag supports MagSafe attachment, allowing visitors to easily affix the device to their smartphones. This feature is designed for ease of use and does not access any personal information from the user's phone.

2.4.1.2 Beacon Installation

- Placement: Install the Beacon at predetermined locations within the museum, noting the exact coordinates for each. The mounting height should be around 6 feet, making sure that each Beacon is mounted at a similar height. Also make sure the antenna is facing out and exposed.
- Power: The Beacons require continuous power and should be connected to a MicroUSB power adapter.
- Beacon ID Configuration: Each Beacon comes pre-programmed with a unique ID, which is also marked on its enclosure. While the Beacons are designed to be plug-and-play, you have the option to modify the Beacon's configuration, including its ID, and programming via the Arduino IDE if necessary.

2.4.1.3 RFID Stickers

- Installation: The RFID Stickers are pre-encoded with unique IDs and are ready for immediate use. Simply affix the stickers around the designated exhibits to integrate them into the MuseumMate system.

By carefully following these instructions, you will successfully set up the MuseumMate system, enhancing the visitor experience at your museum with interactive and personalized tours.

2.4.2 Server

This section outlines the detailed configurations needed for the MuseumMate system's server components. We cover the setup for Node.js with Express.js, InfluxDB for database management, and MinIO for multimedia storage solutions.

2.4.2.1 Node.js and Express.js

- Prerequisites:
  - Node.js (version 14.x or later)
  - NPM (Node Package Manager)
- Set Up
  - Run 'npm install' to ensure all dependencies are available
- How to Start
  - Run 'node index.js' to start the server
- Location Initialization:
  - Must establish floor plan dimensions based on a singular 2D plane like seen below. uwbDevices denotes the location of each beacon in the

exhibit, rooms denotes the diagonal coordinates of each room/exhibit, and preSetGraph denotes the connection points between different rooms.

```
const uwbDevices = {
    "1": { x: 2.2, y: 3.2 },
    "2": { x: 7, y: 3.2},
    "3": { x: 7, y: 0.001 },
    "4": { x: 11.3, y: 3.2 },
    "5": { x: 15.45, y: 3.2 },
    "6": { x: 14.7, y: 8.8 },
    "7": { x: 18.07, y: 14.3 },
    "8": { x: 18.08, y: 18.2 },
    "9": { x: 12.4, y: 18.2 },
    "10": { x: 3, y: 11.4 },
    "11": { x: 0.0001, y: 18.2 },
    "12": { x: 20.9, y: 24.6 },
    "13": { x: 20.9, y: 19.4 },
    "14": { x: 12.9, y: 22.4 },
};

const rooms = {
    "1": { x1: 12.4, y1: 18.2, x2: 20.9, y2: 26.4, maxOccupancy:30},
    "2": { x1: 0.001, y1: 16.2, x2: 12.4, y2: 18.2, maxOccupancy:30},
    "3": { x1: 0.0001, y1: 6.8, x2: 6.5, y2: 16.2 , maxOccupancy:30},
    "4": { x1: 0.00001, y1: 0.000001, x2: 20, y2: 6.8 , maxOccupancy:30},
    "5": { x1: 12.4, y1: 6.3, x2: 20, y2: 18.2, maxOccupancy:30 },
};

const preSetGraph = {
    '1': { '2': 5, '5': 5 },
    '2': { '3': 5, '1': 5 },
    '3': { '2': 5, '4': 5 },
    '4': { '3': 5, '5': 5 },
    '5': { '4': 5, '1': 5 },
};
```

2.4.2.2 InfluxDB

- Installation:
    - Follow the installation guide based on your OS from the Official InfluxDB Documentation.
- Initialize Client in index.js

```
const { InfluxDB } =
require('@influxdata/influxdb-client');

const url = 'http://localhost:8086';
const token = 'YourAuthToken';
const org = 'yourOrg';
const bucket = 'yourBucket';

const client = new InfluxDB({ url, token });
const queryApi = client.getQueryApi(org);
const writeApi = client.getWriteApi(org, bucket);
```

2.4.2.3 MinIO

- Installation:
  - Follow the installation guide based on your OS from the Official MinIO Documentation.
- Client Initialization:

```
const Minio = require('minio');

var minioClient = new Minio.Client({
    endPoint: 'localhost',
    port: 9000,
    useSSL: false,
    accessKey: 'minioadmin',
    secretKey: 'minioadmin'
});
```

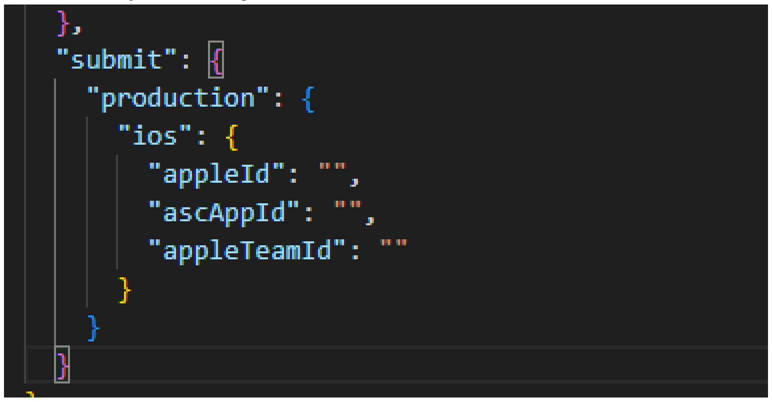
- How to Start
  - Run this command in the directory of your data folder:
    - minio server /data --console-address ":9001"

2.4.3 ChatGPT Server

- Prerequisites
  - Latest version of Golang
  - Gin REST API Framework
  - OpenAI API Key
- Set Up
  - Run ‘go mod tidy’ which installs/updates all dependencies
  - Run ‘go run main.go’ to start the server

2.4.4 Mobile App

- Prerequisites:
  - Node.js (version 14.x or later)
  - NPM (Node Package Manager)
  - Expo CLI
  - TestFlight
  - App Developer Account

    - Expo Developer Account
- Set Up
    - Run 'npm install' to ensure all dependencies are available
    - Check the IP address of your server and replace it in the RFIDScreen.js and CurrentLoc.js
- How to Start
    - For local development: Run 'npx expo start' to start the local development, and scan the QR code in the terminal to open with expo go
- How to Package and Deploy to Test Flight:
    - Enroll in the Apple Developer Program and create an Expo Developer account
    - Install Expo Application services "npm install -g eas-cli"
    - Log into expo account "eas login" in the terminal and enter your credentials
    - Run "eas build:configure" to configure you project and select the desired platform
    - Next, create a build "eas build --platform ios"
        - create a bundle identifier, any string to uniquely identify your app
        - Log in to you Apple Developer Account to automate the majority of the set up process
        - Generate a new Apple Distribution Certificate and Apple Provisioning Profile
    - Go to App Store Connect and create a new app
        - Select platform, a name, a language, select the bundle identifier you created from the drop down, and create a SKU (typically the same string as the bundle ID)
    - Return to your eas.json and follow the code below
    - 

```
},
"submit": {
  "production": {
    "ios": {
      "appleId": "",
      "ascAppId": "",
      "appleTeamId": ""
    }
  }
}
```

        - ascAppID found in app information in App Store Connect
        - teamId found in Apple developer account settings under membership
- Submit to App Store/TestFlight
    - Run "eas submit -p ios --latest"
    - Generate a new ASC Key
    - Login to App Dev account
    - Allow a few minutes to process
- When complete Navigate to TestFlight and see you build
    - For missing Compliance select none of the above

    - Now create your TestGroups for internal testing to invite people to download the app, or send to app for review and allow anyone with the link to download the app under external testing

2.4.5 Dashboard

- Prerequisites:
    - React.js
    - NPM (Node Package Manager)
- Set Up
    - Run ‘npm install’ to ensure all dependencies are available
- How to Start
    - Run ‘npm start’ to start the dashboard

3 Operation of the Project

3.1 Operating Mode 1: Normal Operation

3.1.1 Hardware

Beacons

- Setup beacons in their enclosures according to map
  - If in a new room, take coordinates and map outlets based on predefined coordinates
  - If room is already mapped, place beacons according to outlet coordinates

TourTag

- Connect lithium-ion battery unit to TourTag. Ensure the battery is charged by either charging it beforehand or charging it through the TourTag charger module USB connection.
- Place the TourTag in the enclosure.
- Attach TourTag to mobile device using Magsafe. TourTag can now be used by scanning its QR code through mobile application.

3.1.2 Server

- Node.js and Express.js
  - Run the command: 'node index.js' while in the same directory as the index.js file
    - This will start the server and all corresponding operations
- InfluxDB
  - All client information should already be stored inside of the index.js file
  - Run the command 'influxd' to start InfluxDB
- MinIO
  - All client information should already be stored inside of the index.js file
  - Run the command: minio server /data --console-address ":9001" while in the root directory

3.1.3 ChatGPT Server

- Run the command 'go mod tidy' to ensure all dependencies are installed/updated
- Run the command 'go run main.go' to start the server

3.1.4 Dashboard

- Once packaged and hosted the dashboard should be automatically populated with the current data and be ready to use.

3.1.5 Mobile App

- Once packaged and on test flight the app should be downloadable and launchable with no further issue (for info about packaging refer to 2.4.4)

- Navigate the screens and follow the instruction

3.2 Operating Mode 2: Abnormal Operations

3.2.1 Hardware

Beacon

- System might fail to obtain proper localization information from beacons when beacons are too sparsely populated in an area due to trilateration algorithm requirements
    - In order to ensure trilateration accuracy, plan to place beacons such that each beacon is within range (<=50m, clear line of sight) of at least two other beacons.
- Trilateration accuracy is dependent on static and powered beacons; if beacon is shifted or disconnected from power, erroneous localization might occur
    - Place beacons securely on walls at/above eye level to prevent accidental collisions with beacons
    - If possible, do not sparsely place beacons in order to ensure that trilateration readings are preserved even if a few beacons suddenly lose power

3.2.2 Server

- The server runs through the index.js file, this file is set up to fail gracefully and to preserve data. If errors start to appear in the console, these are the typical reasons:
    - Predetermined data is not populated (uwbDevices, rooms, and preSetGraph) as seen above
        - fill in the missing data and run the server again
    - InfluxDB is not working properly
        - Check to ensure that the client configuration in index.js is correct with your instance
        - Ensure that the InfluxDB is in fact running on the desired port and address
    - MinIO is not working properly
        - Check to ensure that the client configuration in index.js is correct
        - Ensure that MinIO is in fact running on the desired port and address

3.2.3 ChatGPT Server

- The ChatGPT Server runs as a microservice; if not working these are usually the issues
    - OpenAI API Key is not available
        - If the key has been exposed anywhere (Github, etc.) OpenAI disables it
        - Generate a new key on the OpenAI website
    - OpenAI API Key says that "quota is reached"
        - The available queries on a standard trial were reached, purchase more credits or find an alternative option

3.2.4 Dashboard

- If there is an error with the dashboard it usually stems from the queries to InfluxDB. Please check that you are using the correct queries and organization identifiers

3.2.5 Mobile App

- Most errors of the mobile app come at the time of packaging the app here are some common issues and solutions:
    - Incorrect versions of packages
        - Within the Expo Developer platform under builds you should see the logs and the processes that pass. The version issues should be seen "Run expo doctor" with instructions of how to fix these issues.
            - Caution this error should be overlooked do not install the newer versions as they are incompatible with other aspects of the app: `react-native-gesture-handler@2.16.0 - expected version: ~2.12.0 react-native-reanimated@3.8.1 - expected version: ~3.3.0`
- Version issues:
    - After each change the version of the app should be incremented in the app.json to avoid denial by App store connect.

4 Technical Background

4.1 Hardware

4.1.1 Beacons

Beacons, key to the network, are placed at fixed points throughout the museum. These small devices continuously emit UWB signals with unique identifiers, which are sampled and read by TourTags. In combination with the TourTags, the network of beacons allows the system to accurately track and guide users within the museum without requiring personal data.

4.1.2 TourTags

The TourTag, the main user device, is an ergonomically designed tool, tailored for ease of use and interactivity during museum visits. It is housed in a 3D printed enclosure and consists of a PCB containing the following parts:

- ESP32 UWB DW3000: The main processing unit of the TourTag
- RC522 RFID Module: Module for scanning RFID tags
- Adafruit USB Li-ion/Li-poly Charger Module: Manages main power to TourTag and charging capabilities
- Lithium Ion Polymer Battery: Power source
-

The TourTag serves as the target for our user localization service and is designed to typically be attached to a mobile device through the use of Magsafe magnets, but it can be handheld as well.

Connectivity and Data Transmission:
TourTags securely connect to the museum's network using WPA2-PEAP. This connection is crucial for the consistent transmission of data between the device and the server.

RSSI Data Collection and Transmission:
TourTags use UWB technology to obtain precise distance readings, which, along with unique beacon IDs and TourTag IDs, is sent to the server through UDP Port 3333. This data is essential for the system's location tracking capabilities.

RFID Integration and Data Handling:
The RC522 module detects RFID TAG IDs placed near museum exhibits upon close proximity to the tags (~5cm or less). Upon recognition of an RFID TAG, the device sends the scanned ID to the server via UDP Port 3334.

4.2 Server

Our system is a robust and innovative solution designed to enhance the visitor experience within museum environments. At the core, it utilizes a Node.js server that acts as a central processing hub, receiving Ultra-Wideband (UWB) distances and Radio-Frequency Identification (RFID) data via User Datagram Protocol (UDP) from an ESP32 module.

This data is critical in providing real-time location services and interactive content to museum visitors.

UWB and Location Processing:
The server accepts UWB readings from TourTag via UDP and stores them in a data structure that updates when it receives new values. Performs trilateration on the UWB data to determine the x,y coordinates of each user. The x,y coordinates are accurate within 10 cm and used to determine which rom the user is in.

Congestion Management:
The server utilizes a dynamic graph that updates as room occupancy’s change as visitors navigate the museum. This is done by calculating the Euclidean distance between rooms and finding the room occupancy percentage between every edge in the graph and combining the two to establish a new “weight” for each edge in the graph. This graph is used by all routing services.

Routing:
The server utilizes the weighted graph discussed above that is dynamic to the visitors in the museum. This graph is then used to perform Djikstra’s algorithm for single node travel (room to room) and Traveling Salesman Problem (TSP) for multi-node travel (multiple rooms). This optimizes the travel speed of all visitors in the museum and aims to avoid congestion.

RFID Data Handling:
In parallel, the RFID information captured by the ESP32 is sent to the server, which then forwards it through a WebSocket connection to a front-end application built with React Native. This enables the app to provide visitors with immediate multimedia content relevant to their location and the exhibits they are engaging with.

Data Transmission (InfluxDB):
The server collects data such as room occupancy and room stay durations to display on the administrator dashboard. The data is collected and then sent to InfluxDB.

MultiMedia Retrieval (MinIO):
The server retrieves the multimedia files from MinIO, then creates a custom URL for each of them so that they can be retrieved and accessed via the mobile application.

REST API Gateway:
The server is equipped with a REST API gateway that exposes four distinct endpoints, each serving a unique function within the system's ecosystem:

- /location/:userID:
    - This API endpoint retrieves the location of a specific TourTag
- /path/:userID:
    - Leveraging Dijkstra's algorithm, this endpoint calculates the shortest path for a user device from its current location to a designated room. This

feature aids visitors in navigating the museum's layout without unnecessary detours, enhancing the overall visitor experience.

- /tsp-path:
  - Designed to accept an array of room locations, this API employs the Traveling Salesman Problem (TSP) algorithm to determine an optimal path that visits each room once. This is particularly useful for guided tours or exhibits that require visitors to follow a specific sequence.
- /rfid/:bucketName
  - Upon receiving an RFID tag number, this endpoint fetches and returns the corresponding multimedia content stored in MinIO, an object storage service. This allows for scalable and efficient retrieval of rich media resources, delivering a seamless content delivery network that enhances educational and interactive elements for visitors.
- /send-message
  - This endpoint is used to forward messages to the mobile application
- /api/popular-rooms
  - This endpoint returns a ranked list of exhibits by popularity, determined through weighted averaging on important metrics
- /api/exhibit-rating
  - This endpoint is used to forward exhibit ratings to InfluxDB

4.3 Front-End

Upon entering the museum, visitors are greeted by the HomeScreen of the application, which presents a welcoming interface and initiates the engagement process. The user is prompted to scan their TourTag, linking it to the app and activating the navigation features. The BarcodeScanner component takes center stage here, providing a seamless start to the museum exploration.

MuseumMate's interface is meticulously crafted to cater to diverse preferences. Through the TourType screen, visitors can select from various guided tours—each carefully curated to cover specific themes or exhibits within a set timeframe. Alternatively, they can opt for an exploratory mode, enabling a more spontaneous journey through the museum.

For visitors embarking on a timed tour, the PreMade component presents a selection of tours with estimated durations, allowing them to manage their visit according to their schedule. As visitors embark on their chosen path, the CurrentLoc component actively updates their location in real-time, aided by the BLE beacons. The application guides them from one exhibit to the next, ensuring they remain on the most efficient route, all the while avoiding congestion and enhancing their overall experience.

Exhibit Interaction and Multimedia Content Delivery

Each exhibit is equipped with an RFID tag that the TourTag automatically scans when in proximity. This triggers the RFIDScreen component, which fetches and displays detailed multimedia content related to the exhibit. This could include audio guides, high-resolution images, videos, or textual histories, providing a rich, interactive learning experience. The content is dynamically loaded and designed to be engaging and accessible, catering to a broad demographic of museum visitors.

APIs and Websocket connections:
API Testing:

a. /location/:userID (GET)
   i. Retrieves the current location of the user's device and updates the display or UI elements.
b. /path/:userID (GET)
   i. Provides a navigation route for a specified userID to a selected room within the app.
c. /tsp-path (POST)
   i. Receives a user's selected rooms and computes the shortest visitation path.
d. /rfid/:bucketName (GET)
   i. Returns all multimedia associated with a given bucket name from minIO.
e. WebSocket ws://128.197.53.112:8080
   i. Establishes real-time connection for RFID data, updating UI with new object info.
f. Google Translate API https://translation.googleapis.com/language/translate/v2
   i. Translates text to selected user language for improved accessibility.
g. GoLang API http://128.197.53.112:4040/chat (POST)
   i. Accepts prompts and returns informative responses for interactive user engagement.
h. /api/exhibit-rating (POST)
   i. Accepts the rating of an exhibit and posts it to the server which inturn stores the rating in InfluxDB.

4.4 Dashboard:

The admin dashboard serves as a critical tool for museum administrators, providing a comprehensive and intuitive interface for monitoring and managing data. It connects seamlessly with InfluxDB, a time-series database optimized for fast, high-availability storage and retrieval of time-stamped data. This setup enhances operational efficiency and delivers detailed insights into visitor interactions and behaviors, thereby aiding in strategic decision-making and resource allocation.

Performance Specifications

Dashboard Load Time: The dashboard is designed to load completely within 3 seconds after login, ensuring quick access to data without significant wait times.

Data Visualization Rendering: All graphical data representations, including charts, graphs, and maps, are optimized to render within 5 seconds following a data request. This rapid rendering is critical for real-time data analysis and timely decision-making.

Live Data Feed: The dashboard features live data feeds, such as visitor counts and real-time location tracking, which update automatically within a 30-second delay of data reception. This feature ensures that administrators have up-to-the-minute information without needing to manually refresh the dashboard.

Navigation and Responsiveness

Screen Transition Speed: Navigation across different sections of the dashboard, such as data views and settings, occurs within 1 second. This promotes an efficient workflow and minimizes downtime between tasks.

Interactive Elements Response Time: Interactive elements like dropdown menus, sliders, and buttons are highly responsive, with a reaction time of less than 500 milliseconds. This responsiveness is crucial for a smooth and engaging user experience.

User Interface (UI) Consistency and Accessibility

UI Element Consistency: All user interface components, including buttons, icons, and text fields, are consistently designed across various screens. This consistency supports usability and reduces the cognitive load on users as they navigate through different dashboard functionalities.

Integration with InfluxDB

The dashboard's integration with InfluxDB is a key aspect of its functionality. InfluxDB provides efficient handling of time-series data, which is essential for tracking dynamic and time-sensitive museum data such as visitor flows and exhibit interactions. This integration not only facilitates real-time data processing and visualization but also enhances the storage and retrieval of historical data, allowing administrators to perform long-term trend analysis and reporting.

5 Relevant Engineering Standards

Electrical Design and Construction

- IEEE 1547: This standard delineates the requirements for the integration and interoperability of distributed energy resources with electrical power systems. It guarantees that the electrical components within the MuseumMate system, such as TourTag and Beacon devices, meet safety and performance criteria when connected to the electrical grid.
- National Electrical Safety Code (NESC): This code offers a set of guidelines for the safe installation, operation, and maintenance of electrical power and communication utilities. Adherence to this code ensures the safety of MuseumMate's electrical components for both end-users and service personnel.

Software Design and Coding Standards

- MISRA C: Given that the MuseumMate system incorporates embedded system programming, compliance with MISRA C guidelines ensures the reliability, maintainability, and portability of the C code utilized in developing the firmware for TourTag and Beacon devices.
- React Native Framework: Adherence to best practices in React Native ensures that the app is built on a reliable, community-supported framework, optimizing performance and compatibility across both Android and iOS platforms.
- Expo Managed Workflow: Utilizes the managed workflow provided by Expo, which abstracts much of the configuration and lowers the barrier to entry for developers, while ensuring applications are built with industry-standard approaches and tools.

Communication and Internet Protocols

- IEEE 802.11: This series of standards specifies the protocols for wireless local area network (WLAN) communications. Adherence to these standards is essential for the WiFi communication of TourTag devices, ensuring both compatibility and security in wireless interactions.
- ISO/IEC 14443: This standard is pivotal for short-range wireless connectivity, often employed in contactless payment systems, access control, and identification.
- ISO/IEC 24730-5: This standard specifically addresses real-time locating systems (RTLS) using Ultra-Wideband (UWB) technology, emphasizing interoperability and compatibility across different devices and systems.
- IEEE 802.15.4z-2020: An amendment to the IEEE 802.15.4 standard, this update introduces Enhanced Ultra-Wideband (UWB) Physical Layers (PHYs) and associated ranging techniques, crucial for precise indoor positioning and communication in low-rate wireless networks.
- WebSockets for Real-Time Communication: Ensuring that the app can handle real-time data communication efficiently with minimal overhead and latency.
- HTTPS/TLS 1.3: Enforces the latest security standards for transmitting data securely over the Internet, protecting user data from interception and tampering.

Operational Environment and Governmental Requirements

- IEC 61131-3: This standard defines the programming languages for programmable logic controllers (PLCs) in industrial settings, ensuring that the software controlling the MuseumMate system adheres to industrial best practices for safety and reliability.

- ISO/IEC 18305: This standard provides a methodology for testing and evaluating indoor positioning systems, ensuring the performance and reliability of such systems in operational environments.
- FCC Regulations: Adherence to the Federal Communications Commission (FCC) regulations is imperative for devices that emit radio frequencies, such as the TourTag and Beacon. These regulations prevent harmful interference with other electronic devices and ensure compliance with radio frequency exposure limits.

API and Integration Standards

- RESTful API Conventions: Following RESTful best practices for API design, ensuring that the app communicates effectively with back-end services in a stateless, cacheable, and uniform manner.
- JSON:API Specification for APIs: Utilizing JSON:API standard for building APIs in JSON, which simplifies the code, increases efficiency, and facilitates better server-client interactions.

User Interface and Accessibility

- Material Design for Android and Human Interface Guidelines for iOS: Following these design guidelines ensures a native look and feel on each platform, improving user experience and interface intuitiveness.

6 Cost Breakdown

Project Costs for Production of Beta Version (Next Unit after Prototype)				
Item	Quantity	Description	Unit Cost	Extended Cost
TourTag	12		$70.82	$849.84
Beacon	14		$44.23	$619.22
RFID Sticker	30		$2.66	$79.80
			Beta Version-Total Cost	$1,548.86

Include a **budget narrative** paragraph or two if there are things you wish to clarify about the cost breakdown.

7 Appendices

7.1 Appendix A - Specifications

Component	**Specification**
TourTag	Initialization in under 10s after unplug the charging cable
TourTag	Get distance between beacons in under 1s
TourTag	Get RFID reading in under 1s
TourTag	Keep working at least 5 hours without manual intervention
TourTag	Start charging in under 10s after plug in the charging cable
TourTag Enclosure	Size of enclosure within 2 mm
Beacon	Initialization in under 2s after powered
Beacon	Keep running at least 10 hours without manual intervention
Beacon Enclosure	Size of enclosure within 2 mm
Server	User location can be determined in under .05s
Server	Route determination in under .5s
Server	MultiMedia retrieval in under .1s
Server	Data transmission in under .01s
Server	All key services listed above operate under .5s when there are 10,000 users
ChatGPT Server	Cached responses return in under .05s
Mobile App	Initialization to HomeScreen in under 2s
Mobile App	BarcodeScanner activation in under 5s
Mobile App	TourTypes display in under 3s

Mobile App	CurrentLocation map retrieval in under 6s
Mobile App	Screen transitions and interactive response in under 2s
Admin Dashboard	Load Time in under 3s
Admin Dashboard	Data Visualization in under 2s
Admin Dashboard	Live Data Feed update in under 1s delay
Admin Dashboard	Screen Transitions in under 1s
Admin Dashboard	Interactive Elements in under 500ms

7.2 Appendix B – Team Information

John Culley
During the creation of MuseumMate, John was responsible for creating, optimizing, and maintaining the server, InfluxDB, and MinIO. John is currently a senior majoring in Computer Engineering and will be working next year as a Back-End Developer with American Eagle Outfitters.

Kwadwo Osafo
During the creation of MuseumMate, Kwadwo worked extensively on hardware and low-level software, primarily focusing on PCB design and integration, circuit build and hardware coding. Kwadwo is currently a senior majoring in Computer Engineering and is seeking a job to complement his skillset.

Kai Imery
As a member of the MuseumMate project team, Kai was responsible for the creation, maintenance, packaging, and optimization of the React Native mobile application. His efforts ensured the app was successfully published on TestFlight for comprehensive testing. Additionally, Kai developed the administrative dashboard using React.js, streamlining backend processes and enhancing user engagement. Upon graduation, Kai is excited to start his career as a Software Engineer at John Deere, where he will join the IT Development Program. This position will allow him to further hone his technical skills and contribute to innovative solutions in a dynamic industry environment.

Ananth Sanjay
During the process of creating MuseamMate, Ananth was responsible for the creation of the 3D-printed enclosures. He tested out different materials and designs. Ananth is a senior studying computer engineering and currently looking for a job.

Yangyang Zhang
For the MuseumMate project, Yangyang was responsible for creating, maintaining, and optimizing the ESP32 firmware for the BLE and UWB versions of TourTag and Beacon. His efforts ensured the stable operation of the hardware functions and reliable data transmission from the hardware for everything the system needs. After graduation, Yangyang will attend graduate school to continue his education in the field of embedded systems and further hone his technical skills.