The University of Hong Kong

Department of Computer Science

2023-2024
Final Report

Sharing the Past with the Public: Augmented Reality User Experiences at Archaeological Sites

Chau Hiu Man 3035783716

Au Wing Yan 3035783728

Supervisor Dr Choi Yi King

Second Examinator Dr Chim Tai Wing

Date of Submission: April 25, 2024

Abstract

The advent of augmented reality (AR) technology has brought about a revolution in various industries, including tourism, in recent years. Currently, tourism applications are mainly focused on marker-based AR in museums or picture identification to display nearby attraction spots. Our team intends to build an application for iOS and Android phones or tablets that uses augmented reality (AR) as an alternative and portable way of viewing tourist attractions and archaeological information from the Ararat Plain Southeast Archaeological Project (APSAP) led by Dr Cobb from the University of Hong Kong. This application will provide a unique experience of AR navigation through Vedi River Valley, Armenia. Using built-in GPS and magnetometer in smartphones and ViroReact, this application locates the landmarks by coordinate systems transition and projects a three-dimensional model of them onto the physical environment, AR such that the users can interact with them, such as triggering a popup information window that contains the details such as name and description. This paper discusses the background, the techniques applied, the final deliverables, and the challenges encountered in the development.

Acknowledgements

We would like to express our sincere gratitude to our supervisor, Dr. Choi Yi King for the guidance and suggestions throughout the project. Without her continuous guidance, dedicated involvement and support throughout the process, this project could not have progressed smoothly.

My team would also like to thank Dr. Cobb, Peter J. and his team for sharing necessary resources and archaeological expertise regarding their fieldwork in Armenia for this project.

Member Contribution

Table 0 illustrates the work contribution of each team member.

Table 0 Contribution of Members

Front-end mobile application	UI	UI design	Chau Hiu Man, Humen
		UI implementation	Au Wing Yan, Ada Humen
	AR navigation	Setup of connection to built-in GPS and magnetometer	Humen
		AR space and real space alignment	Humen
		AR coordinates updates	Humen
		Computation of AR path rendering	Ada Humen
		Computation of bearing degree in direction indicator	Ada Humen
		AR waypoint selection	Ada
		AR comments	Humen
		3D models creation	Ada
Front-end admin console		UI implementation	Humen
Backend	Database	Setup & Hosting	Humen
		Data structure	Humen
	APIs	CRUD operations	Humen
		Authentication	Humen
		Authorisation	Humen
		Querying	Humen
		File Storage	Humen

Table of Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
APSAP	Ararat Plain Southeast Archaeological Project
AR	Augmented Reality
CRUD	Create, Read, Update, Delete
GCS	Geographical Coordinate System
GPS	Global Positional System
JWT	JSON web tokens
PCS	Projected Coordinate System
POI	Point of Interest
SDK	Software Development Kit
UI	User Interface
UX	User Experience
VPS	Visual Positioning Service

1 Introduction

This section provides the background and rationale, offers an overview and significance of our work toward the findings in Vedi, Armenia and sets out the goals of the project.

1.1 Background: Tourism in Vedi Rive Valley Armenia

During the summer of 2019, a team of researchers and students led by Dr. Peter Cobb excavated ancient human material cultures in Vedi, Armenia, since they carried out the Ararat Plain Southeast Archaeological Project (APSAP) [1]. Vedi is located at the southeast edge of the Ararat Plain. For centuries, this region has been a meeting point for Turkey, Iran (formerly Persia), and Russia. It has always played a significant role in transportation, particularly on the historical Silk Road. Today, Armenia is one of the nations participating in the Belt and Road initiative [1].

Enormous, ruined fortification walls up to four meters high, with a central rectangular defensive tower, are preserved at the Vedi Fortress archaeological site [1]. The APSAP team is currently working on reconstructing these walls. By taking this chance, they would like to encourage the public to visit their archaeological site. However, physical reconstruction is an incremental process in which each excavation can only unearth a thin ground layer, as a result, it is difficult to present the whole buildings or artefacts and physically decorate the sites with posters or boards. It is where augmented reality comes in– bringing the historic sites back to life in the field of tourism while creating an immersive experience.

1.2 Background: Current Navigation Apps Published

It is common to use map apps such as Google Maps by Google and Maps by Apple when navigating using a smartphone. However, navigation is often only possible along public roads with these apps. For instance, these apps do not provide detailed information on premises such as theme parks and campuses. In addition, despite the vast amount of location-based data that Google and Apple have, there are still places that are impossible for the company to map. Though those elusive spots can be captured by satellite and aerial imagery, on Google's and Apple's maps they show up as blank [2].

In contrast, some navigation apps specialised for facilities, such as the Hong Kong Disneyland app [3] and Universal Studios Japan [4], exist. They specialise in information regarding premises; hence navigating in private areas, where guidance is not supported with

general map apps, is possible. However, these apps often only provide 2D maps to the visitor. To effectively read maps, it is essential to possess the skill of shifting from a subjective to an objective perspective and mentally rotating one's viewpoint. However, there are varying levels of proficiency in these abilities among individuals. AR navigation is therefore integrated in this project to lead these people to their destination in a first-person perspective.

The central focus of this project is to provide a comprehensive and interactive guide to the Vedi River Valley site by harnessing the advantages of various types of navigation applications. To fulfil this, this project aims to develop a tourist and AR navigation mobile application as a user-friendly and intuitive platform that can assist users in exploring the Vedi River Valley site.

1.3 Project Objectives

This project aims to develop a cross-platform mobile application that can:

i. Promote and enhance public understanding and appreciation of cultural heritage in Armenia through public education and engagement, including tourism.
ii. Facilitate public education in archaeology and cultural heritage.
iii. Reconstruct and visualize heritages in Vedi Fortress digitally.
iv. Encourage the public to explore the Vedi Fortress by tracking down various inspiring virtual cultural heritages and artefacts.
v. Promote sustainable tourism development in the Armenian Araxes River Valley.

The project was completed in two phases:

Phase 1: Research and development.
Phase 2: Integration of real data sets to the application, performance tuning and debugging.

1.4 Project Contributions

It is hoped that the final product will be able to create an immersive experience for tourists, schoolchildren, and other members of the public to visit and learn about the sites with additional information and context about the history.

1.5 Outline of the Report

This report proceeds as follows. Section 2 describes the methodology involved in this project. Development tools and techniques that are applied will be discussed. Section 3 demonstrates the development configuration, interim outcomes and results achieved in the project. Section 4 discusses the difficulties and limitations encountered throughout the development stage. Section 5 states the future work that will be conducted before the release of the app in July 2024. Section 6 summarises this report by providing a conclusion that restates the objectives, highlights outcomes, and presents the progress of this project.

2 Methodology

This section focuses on the techniques and development tools used during app creation, and the reasons for using these tools by analysing their advantages or comparing their players in the same industry. Additionally, this section elaborates on the system architecture implemented in this project.

2.1 Knowledge, Techniques and Functions Involved

2.1.1 Map Services and Navigation Methods

This project attempts to conduct tests outdoors and develop an app that guides tourists around archaeological sites in the Vedi River Valley. Geographical data required by location-based AR greatly relies on the GPS receiver, a magnetometer (electronic compass) and a camera in devices, and hence the operation devices should have spatial awareness, and the ability to scan their surroundings. It was noted that the accuracy of GPS varies depending on the environment in which it is used. Therefore, Visual Positioning Service (VPS) employing machine learning and imagery from mobile device cameras were also researched. However, these techniques, including arranging markers in various places and recording landmark information from diverse environmental viewpoints, take time and effort. Vision-based navigation methods use cameras to capture images of the environment and utilise computer vision algorithms to infer the position and orientation of the device. These methods typically involve feature extraction, matching and recognition, camera calibration and visual geometric algorithms, which require enormous computational resources and algorithm complexity. The heavy computation load required will result in a slow system response. Since this project is designed for use in areas with good GPS reception, such as open areas and universities, GPS and magnetometers are used for AR navigation. In the best situation, it is possible to obtain location accuracy within about five meters of smartphones [5]. Collected data from these sensors is then processed by the system to align the AR space with real space and calculate the distance and direction between users and targets [6], as Fig. 1 illustrates.

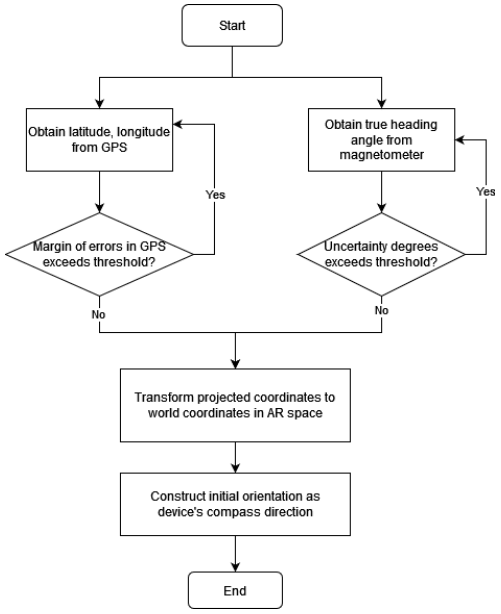


Fig. 1 The flowchart of AR space initialization and alignment with real space processes

A magnetometer is used for matching azimuth, but it cannot be perfectly matched due to unavoidable varying magnetic fields and the user's motion. To minimize the angle offset, this project sets the initial compass heading accuracy threshold as the highest such that the degree uncertainty was kept below twenty degrees. In addition, a low-pass filter algorithm, which will be explained in subsection 2.1.3, is the current workaround to reduce the impact of noise, including motion and magnetic fields varying.

2.1.2 Marker-less Augmented Reality

Early AR technologies were based on markers, an interactive experience activated by a specific image or graphic signal. This process is known as image tracking and recognition and the technique is called marker-based AR [7]. Marker-less AR, on the other hand, does not rely on physical markers like a QR code or image and offers a more immersive experience through real-time scanning of the world environment and simulation of AR objects [7]. In this project, projection-based AR was experimented with, and location-based AR was implemented.

2.1.2.1 Projection-based AR

Projection-based AR relies on the projectors to display digital content onto a flat two-dimensional surface, such as a wall, floor, or paper. It requires *plan detection* which finds horizontal or vertical flat surfaces in the physical surroundings by accessing and using the device's camera for detection [8]. After plan detection, a mesh is projected on the planes and the three-dimensional AR object is placed on the virtual point on the mesh touched by the

users. This procedure is called *object placement.* To increase the realism of the digital content, *light estimation* can be involved to detect real-world lighting by the camera in the devices. The object can then be lit by directional light in the environment. It was noted that light detections depend on the device's hardware, meaning not all devices are available with this function. The program should be able to track and recognize hand gestures because most interactive functions such as object scaling, collection and placement rely on hand gestures such as pinching with two fingers and touching by pointing the index finger.

2.1.2.2 Location-based AR

Location-based AR uses geographical data retrieved by GPS in mobile devices [7]. The program then detects and tracks the user's environment and delivers virtual content at specific locations given the data. Location-based AR relies on real-time GPS tracking to collect geographical data, also called geolocation, for users, while data for digital objects is stored in a database. Geolocation consists of latitude, longitude, and altitude. The GPS also provides a radius of uncertainty in meters in the user's geolocation. In real-time syncing, the user's location also contains a compass direction with magnetic and true heading angles and respective uncertainty degrees.

Matching of AR Space and Real Space

Before starting AR navigation, the AR space must be aligned with the real space. The coordinate system in the AR space uses a right-hand arrangement as it does in ViroReact in which the x-axis is lateral sway, the y-axis is vertical heave (gravitational up), and the z-axis is longitudinal surge [9]. The direction of view is along the negative z-axis. Fig. 2 shows the right-handed coordinate system adopted by ViroReact. By default, the AR space takes the device's location as the origin positioned at $[0,0,0]$ and the device's orientation as the north in which the unit vector is $[0,0,-1]$. The matching operation takes latitude, longitude, and the true heading angle as inputs.

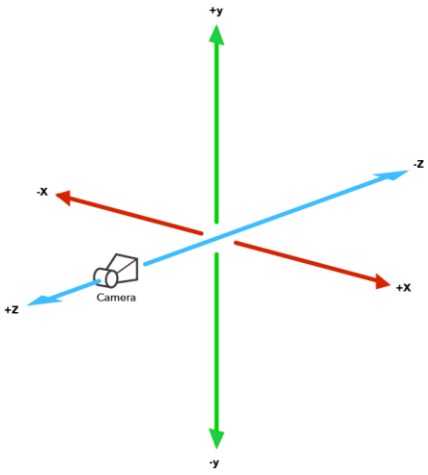


Fig. 2 The 3D coordinate system in the AR space

The procedure can be divided into two phases: Mercator projection which converts the spherical Earth with latitude and longitude into cylinder coordinates, and conversion from the Mercator projected coordinates into world coordinates in AR space, which consists of the translation of origin and matching of compass direction to device orientation in AR space. The system uses the **WGS84 standard** for the geographical coordinate system (GCS), and **Web Mercator** for the projected coordinate system (PCS). The Mercator projection follows. Let rad denote the formula of converting degree to radian as seen in (1), R be the constant radius of Earth in meters, z be the latitude and x be the longitude. Equations (2) and (3) show the Mercator projection of latitude and longitude respectively.

$$rad(\theta) = \frac{\theta}{180} \times \pi \tag{1}$$

$$x' = R \cdot rad(x) \tag{2}$$

$$z' = R \cdot \log\left(\frac{\sin\big(rad(z)\big) + 1}{\cos(rad(z))}\right) \tag{3}$$

Step two is the conversion from the projected coordinate to the world coordinate in AR space. Let $\vec{u} = (x_1, z_1, \theta_0)$ be the user's coordination where θ_0 is the initial true compass heading in degree, in other words, the orientation relative to the y-axis, and $\vec{v}_{global} = (x_2, z_2)$ be the AR object's coordinate in the xz plane of the global space. Their coordinates are computed by equations (2) and (3) given their latitude and longitude. The map is adjusted to be user-centred by subtracting these two projected coordinates, as shown in the change from Fig. 3 (a) to (b). The z-axis is flipped by multiplying -1 as the AR space has the negative z-axis facing north. Automatic adjustment that aligns a certain azimuth with a magnetometer is then

obtained by the counterclockwise rotation around the y-axis by θ_0 (see Fig.3 (c)). The overall transformation can be expressed mathematically as:

$$\vec{v}_{local} = R \times (F \times (\vec{v}_{global} - \vec{u})) \quad (4)$$

where R is a 3×3 rotation matrix by the angle θ'_0 about the y-axis:

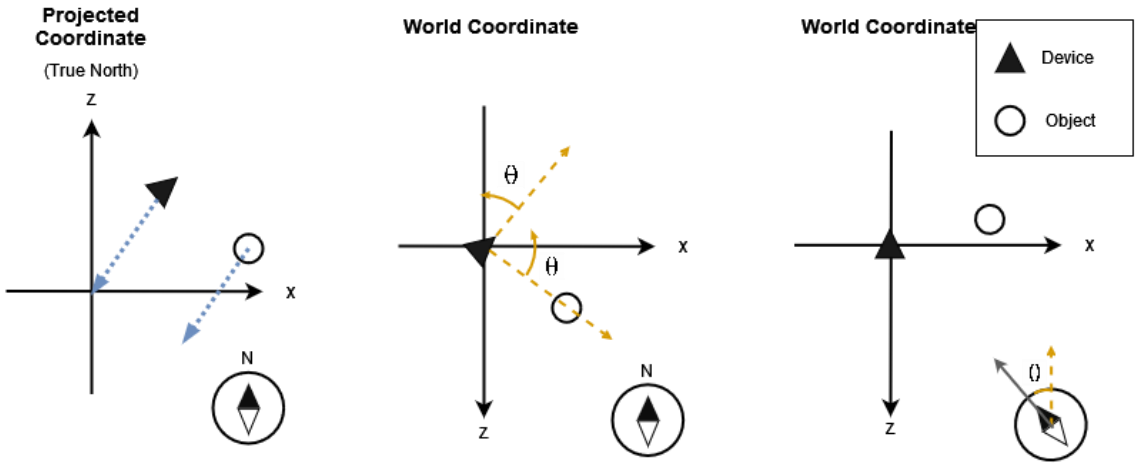
$$R = \begin{bmatrix} \cos\theta'_0 & 0 & \sin\theta'_0 \\ 0 & 1 & 0 \\ -\sin\theta'_0 & 0 & \cos\theta'_0 \end{bmatrix}$$

F is a 3×3 matrix that flips the value of the z-coordinate by -1:

$$F = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

θ'_0 is the device's initial true compass heading in radian computed by the equation (1), and the remaining variables were computed as:

$$\vec{u} = [x_1, 0, z_1]^T$$
$$\vec{v}_{global} = [x_2, 0, z_2]^T$$
$$\vec{v}_{local} = [x'_2, 0, z'_2]^T$$



(a) Coordinates in the Mercator projection coordinate

(b) Translated coordinates in the AR space

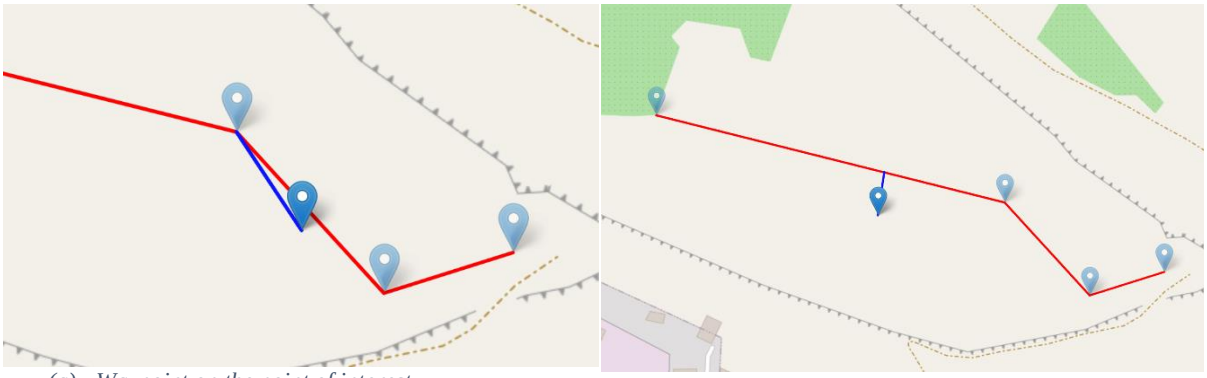
(c) Final rotated coordinates in the AR space

Fig. 3 Conversion of the object coordinates from the projected coordinates to the world coordinates

AR Navigation to Waypoint

Once the user selects a route and a point of interest (POI), navigation objects are placed along the route. These objects have their coordinates computed to determine their location. There are two types of navigation objects: a route object showing walking directions and waypoint

objects located at the POI and the nearest location between the user and the route. In the example shown in Fig. 4(a) and (b), the POIs are represented by pale blue markers and form a route indicated by a red line. The blue marker represents the user, and the waypoint is the intersection of the blue path and the red route. If the distance between the user and the POI is more than 25 meters, the system will calculate the closest point on the route relative to the user and render it as a waypoint, as shown in Fig. 4(b).



(a) Waypoint on the point of interest

(b) Waypoint on the nearest point between the user and the route

Fig. 4 Waypoint selection on the route in 2D map view

The *orthogonal projection* of a point onto a line can be used to obtain the closest point on the route as shown in equation (5). Let n be the number of attractions on a route, p_{i-1} and p_i be two consecutive points to form a straight line in which p_{i-1} represents the last attraction the user visited and p_i represents the POI and i is limited to between 0 and $n-1$, and the user coordinates be u. Denote the coordinates of the waypoint as q, the vector obtained by subtracting u by p_{i-1} as $\overrightarrow{u'}$, the straight line between p_{i-1} and p_i as $\vec{v}$.

$$q = \begin{cases} p_i, & \text{if } i = 0 \text{ or } d \leq 25; \\ p_{i-1} + t\vec{v}, & \text{otherwise} \end{cases} \tag{5}$$

where d is the *latitude and longitude distance in meters* between u and p_i:

$$d = 2 \cdot R \cdot atan2(\sqrt{a}, \sqrt{a-1}) \tag{6}$$

in which R is the radius of the Earth in meters, function $atan2(y, x)$ is:

$$atan2(y,x) = \begin{cases} \tan^{-1}(\frac{y}{x}), & x > 0, \\ \tan^{-1}\left(\frac{y}{x}\right) + \pi, & x < 0 \text{ and } y \geq 0, \\ \tan^{-1}(\frac{y}{x}) - \pi, & x < 0 \text{ and } y < 0, \\ +\frac{\pi}{2}, & x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2}, & x = 0 \text{ and } y < 0, \\ 0, & x = 0 \text{ and } y = 0. \end{cases}$$

a is obtained by taking the latitude and longitude of u and p_i, which are expressed as lat and lon in radian respectively:

$$a = \sin^2\left(\frac{p_i.lat - u.lat}{2}\right) + \cos(u.lat) \cdot \cos(p_i.lat) \cdot \sin^2\left(\frac{p_i.lon - u.lon}{2}\right)$$

and t is computed as:

$$t = \max(1, \min(0, \frac{\vec{v} \cdot \overrightarrow{u'}}{|\vec{v}|^2}))$$

Updating Coordinates of AR Waypoints

Over time, two coordinate systems might start to fall out of synchronisation. Since the system adopts location-based, also known as sensor-based, navigation primarily relying on built-in GPS and magnetometer to acquire the user's positional and directional information, the approach to avoid out-of-syncing is updating the coordinates of AR waypoints once receiving the high-precision geographical position. The definition of high-precision geographical position in this project refers to the geolocation with a radius of uncertainty for the location less than or equal to 5 meters. The cycle of obtaining GPS data is set to 1 second. Considering a trade-off between GPS accuracy and real-time response, which necessity will be explained in sub-section 4.1, the uncertainty radius will be increased to 10 meters after 10 seconds when there is no high-precision location update. Fig. 5 illustrates the flow of updating coordinates after obtaining geographical coordinates from the device's GPS.

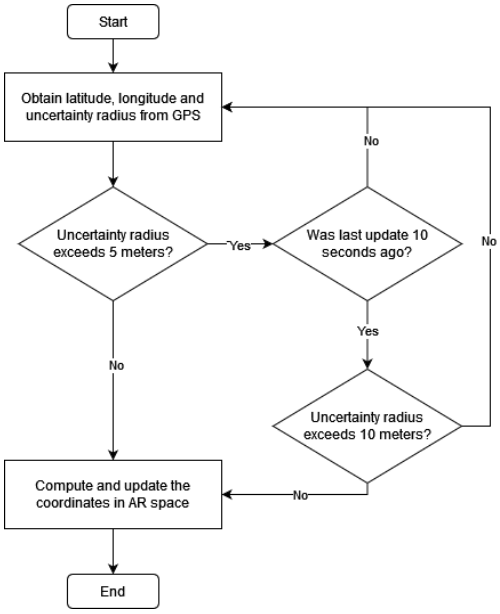


Fig. 5 The flowchart of updating coordinates

The update of AR waypoints' coordinates involves the computation of the local coordinates relative to the user and transforming the coordinates based on the current camera's position. It was observed that the shortcoming of this approach is the instability of rendering positions of the virtual spatial objects, that is the AR waypoints might suffer from jitter, drift and other phenomena which affect the user experience. Nonetheless, the synchronisation between the GCS and the PCS returns a better global position alignment compared with only depending on the camera's motion tracking after the initial AR space matching.

Heading Angles Computation

The AR navigation should be able to direct the user to the target with a device location and orientation. Let deg denote the function of converting radian into degree. Given two geographical coordinates p_1 and p_2, the bearing degree β measured in the clockwise direction from the north line with p_1 as the origin to p_2 can be obtained by the following equation:

$$\beta = 360 - ((\deg(rad(atan2(y, x)) + 360) \,\%\, 360) \tag{7}$$

where x and y are computed as:

$$x = \cos(p_1.lat) \cdot \sin(p_2.lat) - \sin(p_1.lat) \cdot \cos(p_2.lat) \cdot \cos(p_2.lon - p_1.lon)$$

$$y = \sin(p_2.lon - p_1.lon) \cdot \cos(p_2.lat)$$

2.1.3 Low-pass Filter Algorithm

A low-pass filter algorithm, which reduces the high-frequency content of a signal while allowing lower frequencies to pass through [10], is applied to reduce the noise in obtaining the heading angle from the built-in magnetometer. The low-pass filter applied in this project used an array with a maximum size of 10 to store the previous inputs and current outputs. Let n be the current size of the array and φ be the smoothing value with a range of [0, 1). Default value of φ is 0.5. As φ goes toward 0, the filtering effect becomes weaker. The difference equation for the low-pass filter is:

$$y_0 = \begin{cases} x_0, & \text{if } n = 10; \\ 0, & \text{otherwise} \end{cases}$$

$$y_i = \varphi \cdot x_i + (1 - \varphi) \cdot y_{i-1}, \qquad 1 \leq i < n$$

where x_i is the value of the ith elements in the array, y_i is the current output sample, and y_{i-1} is the previous output sample.

The first element will be popped if the value of n reaches 10, and the final output y_{n-1} will be appended to the array as an input sample for the next round of computation. This averaging will not alter the input signal much at all if the signal is changing gradually from one sample to the next, but it will smooth the signal considerably if the input changes drastically.

2.2 System Architecture

This section further outlines the system configuration and architecture implemented in this project. Fig. 6 illustrates the components and flow of communication between components in the program. Subsequent sub-sections will present deeper explanations and details on each component.

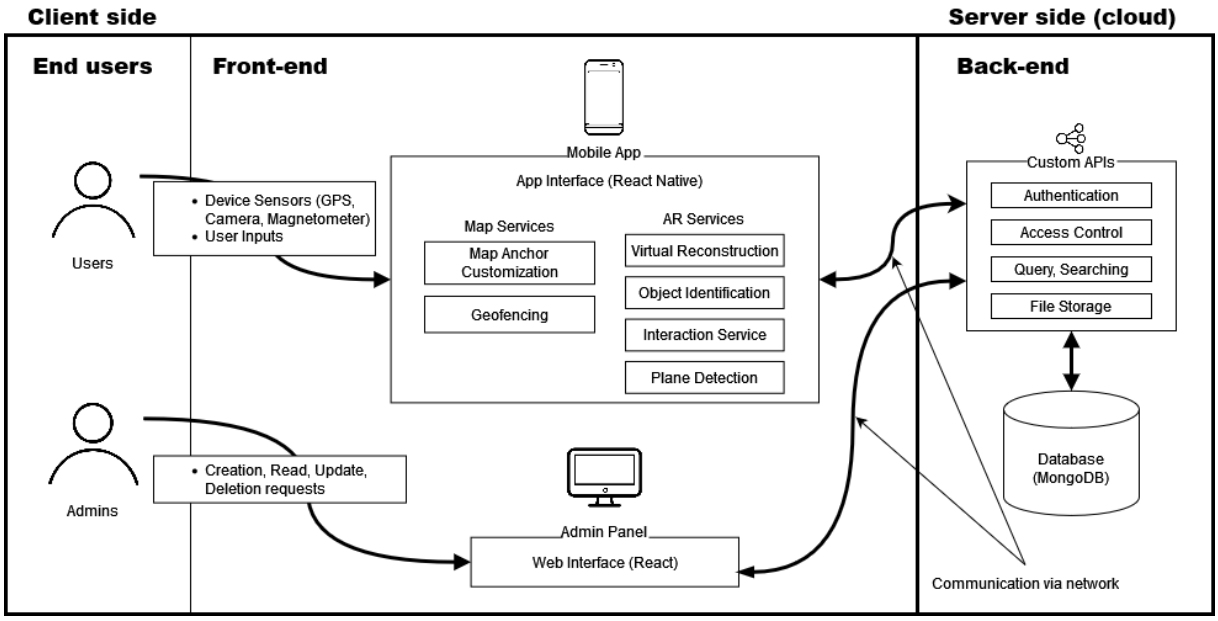


Fig. 6 The system architecture diagram

2.2.1 Front-end Application and AR Services

The front end of the application has a few significant roles. Its primary function is to serve as an interface for users. This interface includes viewing and entering data into the application. Furthermore, the front end is responsible for collecting user personal data and sending it to the backend for storage in the database if necessary. As illustrated in Fig. 6, the personal data is collected in the front end and will be transferred to the backend for storage and further processing. Most importantly, the front end integrates with the AR service such that the users can smoothly go to their destination by receiving AR navigation suggestions.

The front-end service was developed using **Expo**, which is an open-source framework, as well as a bundle of tools and services, for building **React Native** applications [11]. During the decision-making process to determine the appropriate front-end development tools to be utilized, multiple factors were considered. Firstly, React Native is a cross-platform tool that can run applications on both Android and iOS in a single codebase. This streamlines the development process as it significantly reduces programming time compared to developing platforms specifically. Secondly, React Native provides a native rendering engine which ensures consistency in design across different development environments. Lastly, Expo offers EAS build which is a hosted service for building applications for React Native projects [12]. Since this project is in collaboration with the APSAP team which has little experience in

developing software, sharing builds with them, rather than internal distribution such as using ad hoc and/or enterprise “universal” providing, is easier in delivering testing prototypes to the internal team members. This, in turn, facilitates communication and makes it easier to understand the requirements of the APSAP team.

Table 1 Comparison of two popular front-end frameworks

	React Native	**Flutter**
App Size	Medium	Larger
Performance	Worse	Better
External AR Library Supports	More	Fewer
Documentation	Community-driven	Clear and solid
Hot Reload and Reload	Offers reload which refreshes the entire app	Offers reload and hot reload which refresh only updated code
Supported Platforms	Android + iOS	Android + iOS

It is noted that another framework like Flutter, an open-source framework by Google for building multi-platform applications from a single codebase [13], shares similar advantages that React Native provides as previously mentioned. Table 1 compares React Native with its biggest competitor Flutter. In the performance aspect, Flutter even outstands React Native because it eliminates the additional JavaScript layers to run hybrid applications which increases the compiling time in React Native [14]. Furthermore, Flutter provides a hot reload feature which allows the developers to rapidly make code changes and view the result almost immediately without restarting the app, resulting in quicker development. Nevertheless, React Native has a significant advantage in outcompeting Flutter or native codes (e.g., iOS using Swift and Android using Kotlin) -- relatively mature development and external library support in the AR field.

As mentioned in sub-section 2.1.1.2, **ViroReact**, which is a platform for developers to build native cross-platform AR and VR applications using React Native rapidly [15], was used in this project. ViroReact plays a crucial role in AR development in this project because it

eliminates the need for separate AR development in Apple's ARKit and Google's ARCore [15], ultimately speeding up the development process. Although some comments on the online forum suggest that another alternative – AR Foundation by Unity and Vuforia – may deliver better performance in AR rendering and physical engine [16, 17], the project team considered various constraints before opting for ViroReact. Firstly, React Native and Unity have their respective set of tools for building UI components, handling user input and managing state, making integration of Unity in a React Native project challenging and inconsistent in the front-end design. Secondly, given that Vuforia and Unity are built on different programming languages (C# and C++), additional bridging code is required to connect the various parts of the application when integrating them with React Native, which is built on TypeScript in this project. In summary, integrating Vuforia and Unity in a React Native application is not a common and recommended approach.

2.2.2 Front-end Admin Console

Another front-end component in the system architecture diagram shown in Fig. 6 is a web application that serves as the admin control panel. The admin control panel is used for managing the digital content shown in the mobile application, involving the database CRUD operations consisting of the "create", "delete", "update" and "patch" actions.

React was used during the front-end development of the admin console because of its fast rendering time, ease of learning and more stable code structure. In addition, since the frontend mobile application was built with React Native, using React ensures consistency in the frontend code.

2.2.3 Database Management System and Backend

2.2.3.1 Database

Considering the enormous amount of data including but not limited to 3D models, 2D images and pieces of literature on items and geographical sites being stored in a database, **MongoDB**, a NoSQL database, was used. According to the nature of this application, the server heavily depends on querying a database, instead of data calculation and analysis. Real-time calculations, such as the AR object placement and the distance between the user and targets during the navigation, are only limited to front-end processing and, in turn, will not be stored in the database. As a result, customised API endpoints should not be difficult, and the backend was assumed to be lightweight.

This project prioritised data storage. Compared to Firebase, which is another popular NoSQL database known for its ease of learning and integration of backend services, MongoDB outweighs Firebase for its high performance in manipulating massive data and data querying [18, 19, 20]. In addition, MongoDB, as a NoSQL database, offers data in JSON format that helps in faster data interchange and web service results. It also provides higher scalability in data structures and interactions compared to an SQL database, which is beneficial to the project because the scope of the deliverable has undergone changes along the timeline. For the ease of maintaining, organizing, and reading, **Mongoose**, an extension of MongoDB, was used for data schema construction.

2.2.3.2 File Storage Service

MongoDB offers **GridFS** for *file storage*, including storing and retrieving 2D images and 3D models. GridFS enables the storage and retrieval of large documents in BSON format, which is a binary representation of JSON [21]. BSON format is lightweight, traversable, and can handle additional data types such as Decimal128 for converting media.

In a typical approach, a third-party asset storage service like Amazon AWS S3 is utilized in addition to MongoDB to solve the size limitation of an object. However, additional budgets and credentials are required to register those asset storage services. Also, additional client code is required to connect to third-party APIs within the client code. Given the limited budgets for this project and the lack of a universal resource locator (i.e. an HTTP/HTTPS URL) for the APSAP team's data, GridFS was considered the best approach.

GridFS relies on two collections to store files. One collection stores the binary chunks of the files and another stores file metadata as shown in Fig. 7.

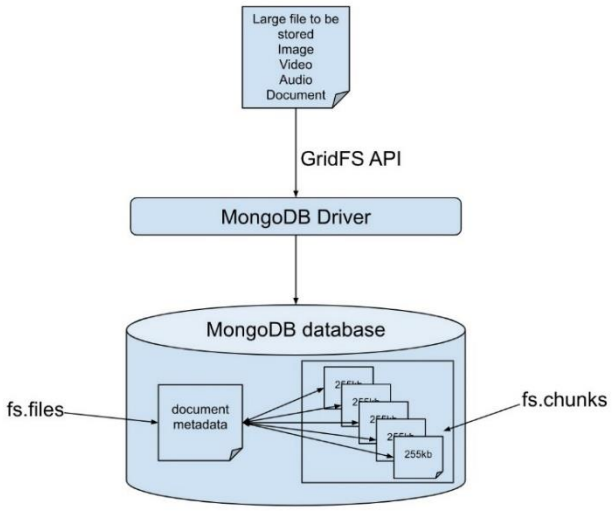


Fig. 7 Flow and structures of uploading large files to MongoDB using GridFS

2.2.3.3 API Framework

Feathers, which is an API and real-time application framework with TypeScript or JavaScript [22], was implemented during the backend development. It supports both MongoDB and frontend technologies involved in this project. A Feathers database adapter for Mongoose called **feathers-mongoose** is utilized to automatically generate services that implement standard CRUD functionality for the data structures given their schemas, resulting in a reduction in code repetition to achieve common service methods on a database. It supports querying, sorting, limiting, and selecting with Feathers query syntax and allows MongoDB basic regex searching and full text searching in `` `find` `` and `` `get` `` method calls, as a result, the development process is simplified. Querying also applies `` `update` ``, `` `patch` `` and `` `remove` `` method calls depending on the need of requests. Another Feathers benefit is that it is a fully compatible and lightweight wrapper over Express.js and Socket.io, allowing the development of RESTful APIs and WebSocket APIs. The formats of parameters in each service method are listed below:

Table 2 Service methods with parameters format in Feathers

Service methods	Parameters	Description
`FIND`	`params`	Retrieves a list or a pagination object of all resources that match the query criteria in `` `params.query` `` from the service.

GET	`id, params`	Retrieves a single resource with the given `` `id` `` from the service
CREATE	`data, params`	Creates a new resource with `` `data` ``. The method returns with the newly created data.
UPDATE	`id, data, params`	Replaces the resource identified by `` `id` `` with `data`. The method returns with the complete, updated resource data. `` `id` `` can be null when updating multiple records.
PATCH	`id, data, params`	Merges the existing data of the resource identified by `` `id` `` with new `` `data` ``. `` `id` `` can be null indicating that multiple resources should be patched with `` `params.query` `` containing the query criteria
REMOVE	`id, params`	Removes the resource with `` `id` ``. The method returns with the removed data. `` `id` `` can be null, which indicates the deletion of multiple resources.

Additionally, Feathers provides a set of standard error classes and customization of error responses, which is beneficial for debugging and front-end bug reports. Meanwhile, Feathers provides powerful pluggable middleware functions called hooks which can be registered around, before, after or on the error of a service method. Permissions, validation, logging, authentication, data validation and resolvers, sending notifications and more can be easily handled by hooks.

Various mechanisms of authentication are supported by Feathers, including local email and password authentication, JWT authentication and OAuth logins such as Facebook, Twitter, etc. At the same time, authorization can be implemented easily with Feathers' hooks. In this project, local and JWT authentications are applied. For security, the HS256 algorithm is implemented for the JWT generation and signing, while salted hashing is applied for passwords in account creation.

3 Results

This section briefly introduces the development environment or configuration and focuses on the details of final deliverables including the mobile application, web-based admin console and backend.

3.1 Development Environment

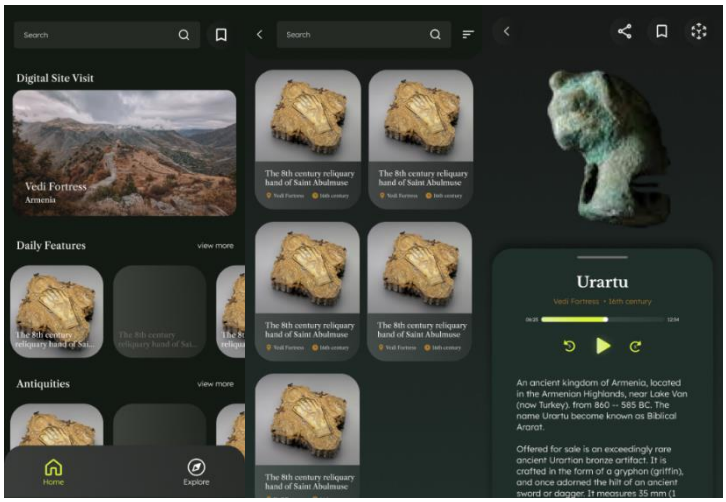
For the development of the mobile application and admin console, a machine with an Apple M1 chip and 16 GB memory was used as the configuration. All experiments related to the mobile application in this report were conducted on a Galaxy S10plus Android smartphone with an Android version of 12, storage of 512GB, and a memory of 8GB.

3.2 Interim Outcomes and Updates Since Mid-term Reviews

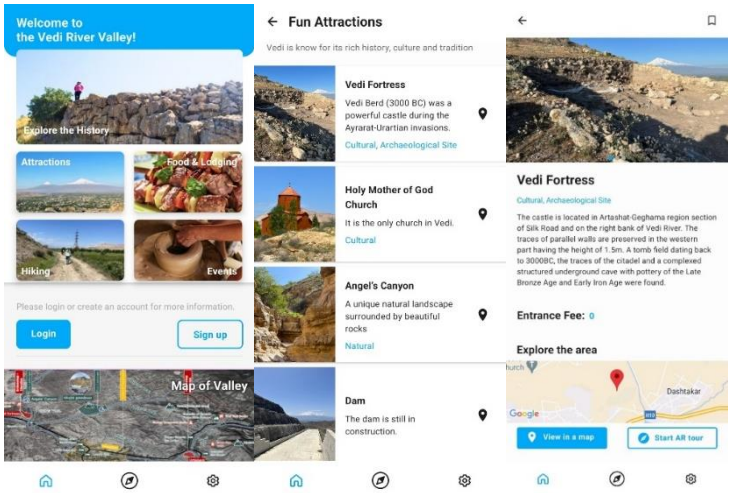
This sub-section describes the interim outcomes produced before the significant review from the APSAP team and outlines the differences between initial and final implementations as following sub-sections after receiving reviews.

3.2.1 Shift in Application Focus: From Education to Tourism

The focus of the application shifted from providing education to catering to tourism, with an expanded geographical area that includes the Vedi River Valley in addition to the Vedi Fortress. As a result, the UI was redesigned, and the backend was revamped.



(a) The initial UI design of the application. The screens from left to right are as follows: Home, Seach Results, Artefact Detail

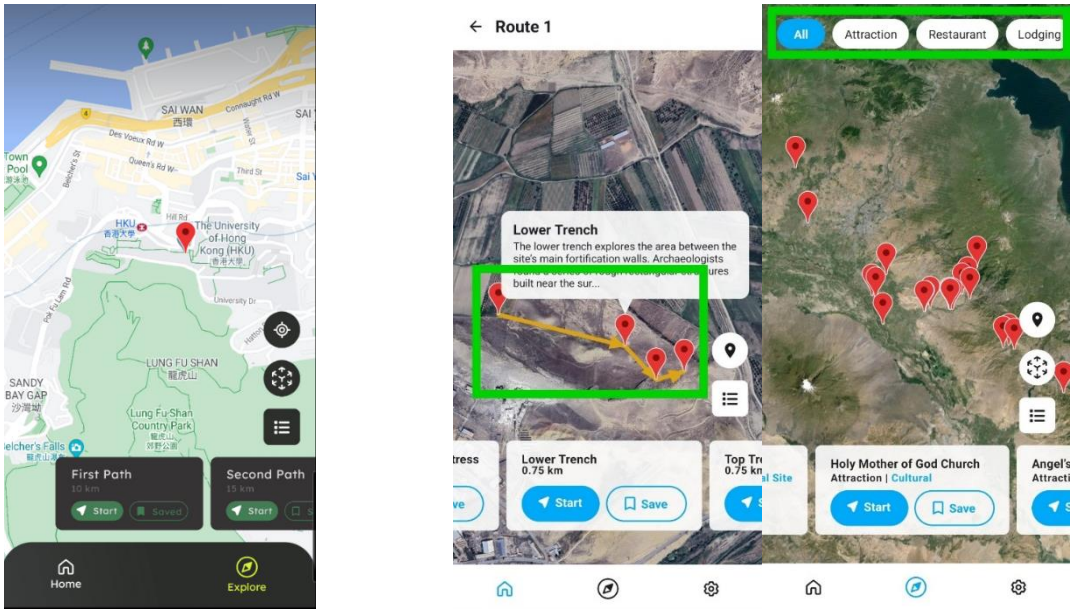


(b) The final UI design of the application. The screens from left to right are as follows: Home, Attractions, Attraction Detail

Fig. 8 The UI designs of the mobile application

The initial UI design of the application is shown in Fig.8(a), while the final UI design is shown in Fig.8(b). As can be seen, the final implementation of the front end differs markedly from the initial design. Apart from the obvious changes in layout and colours, the main content of the application changed from artefacts to tourist spots, hiking routes, and local activities. A scroll view containing different categories of artefacts was replaced with a one-page layout with three separate sections on the home page, as shown in the leftmost of Fig. 8(a) and (b). The attractions page replaced the initial search result page with a grid view of the brief information on the artefacts, shown at the centre of Fig.8(b) and (a), respectively. 3D demonstration and interactions, including zoom-in and zoom-out and rotation with the artefacts, and audio description with text-to-speech function, were also removed from the detail page.

Apart from the pages showing information on tourist attractions, the map page, which helps the user to obtain details of the geographical sites and acts as a starting point in the flow of AR navigation, was improved by dividing it into two separate pages. The left screen in Fig.9 (b) shows the new design that offers a hiking route with a visiting order of tourist sites. A tab bar for filtering types of tourist sites in addition to each location represented as a single red pin on the map is also implemented in the new design of the map page, as shown on the right side of Fig. 9(b).



(a) The initial UI design of the Map Page

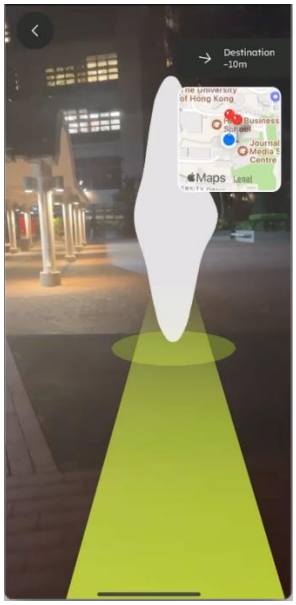
(b) The final implementation of the Map pages. The left screen is the map view of a hiking route. The right screen is the entire map of the Vedi River Valley. The differences with the initial design are highlighted by green rectangles.

Fig. 9 Comparisons of the Map pages in the initial and final version

During the mid-term review, the APSAP team expressed interest in the potential of booking and payment systems. As a result, pages for personal scheduling, booking and payment, in which the details will be illustrated in later subsection 3.3.1, are created.

3.2.2 Improved Navigation Functionality: Multi-Point Guidance

While the core functionality of AR navigation was unchanged, the target and user flow in navigation changed. Specifically, navigation improves from only being able to guide the user to a single POI to being compatible with guiding the users to travel to multiple POIs on a route in order. Fig. 10 demonstrates the comparisons between the initial and updated versions of the AR navigation screen. Before the mid-term review, the user could only be led to a single destination, and no notification or directions would be shown on the screen after the user had reached the destination. The current application has improved the user experience by adding these missing components to the AR navigation.



(a) The initial AR navigation running on an iOS device

(b) The updated AR navigation running an Android device. The updates were circled by red rectangles

Fig. 10 Comparisons of AR navigation screens in the initial and updated versions

3.2.3 Admin Console Development and Authorisation for Data Management

In the past, developers were solely responsible for managing data. It was believed that the APSAP team would not frequently update information related to their findings on artefacts. Furthermore, since this project was being worked on by a small team, it was easier to manually upload data to the database by the developers instead of creating an admin control panel. After the mid-term review, the APSAP team noticed a shift in focus from artefact studies to tourism information. This requires more frequent and dynamic updates to the content. As a result, team members have been granted database access for reading and writing purposes. For their convenience in usage, an admin console panel was built.

At present, the admins of the application are the developers and the APSAP team, so the website deployment and domain registration are not essential during the development phase. However, during the mid-term review, the APSAP team mentioned the potential of local businesses and museums in Armenia to become future collaborators. In that light, role-based access controls were preferably implemented. The details of the types of roles and their respective authorisation will be discussed in a later subsection.

3.2.4 Backend Revamp in API Framework

For better accommodation with the latest functionalities and to solve the problems with the MongoDB Altas backend configuration, the project team decided to switch to the Feathers

API framework from Realm by MongoDB. The differences between Feathers and Realm in various aspects are summarised in Table 3.

Table 3 Comparisons between Feathers and Realm by MongoDB

	Feathers	Realm
Restrictions on Model Classes	Less	More
Customised Errors	More supports	Fewer supports
Third-party APIs Integration	Fewer limitations	More limitations
External Library Supports	More	Fewer
Programming Environment	Local code editor	Cloud MongoDB Altas
Documents	Official and clear	Community-driven

After considering the constraints mentioned in the table, the team chose to make the change for the following reasons:

Integration Challenges with File Storage

GridFS, mentioned in subsection 2.2.3, is the file storage system in this project and is supported by MongoDB. Typically, it is used in node.js contexts and relies on two collections to store files, as explained in subsection 2.2.3. However, implementing this structure in Realm can be difficult due to the platform's model class limitations. Furthermore, MongoDB Realm does not provide the functionality to compute write URLs to upload the asset data and previews of the images and videos. Additionally, the MongoDB Atlas free account has a size limitation. Storing large media files such as images and videos directly in Realm may pose a concern for storage limitation. More importantly, Realm is a full-sync database. The large number of media stored in Realm might overwhelm the user’s device quickly because Realm stores the media data both on the device and in the online storage.

Limitations in developing APIs

The mid-term review suggested a booking and payment system might be necessary for a tourism application. Yet, both systems require the ability of real-time syncing and specialised error handling. For instance, the booking system needs to check and report the user's booking state before proceeding; the payment system should have multiple APIs dependent on the applied payment platforms, such as Stripe, Google Pay, credit card, etc. Implementing such complex datasets and data interactions can be challenging using the functions and triggers provided by Realm.

Restriction on User class in the aspect of data structure

It is important to note that customising user properties in Realm is restricted to a data field named "`custom_data`," which may not be ideal for all applications. Additionally, authorisation is limited to the default User class, as Realm does not support unlimited user classes. To implement multiple user roles, the only solution in Realm is to add a data field to specify it. This restriction makes it more challenging to design multiple user roles and can lead to extraneous data characteristics and constraints. For example, the app users' contact number is unrelated to admin registrations.

Limited Debugging Abilities

Realm lacks easy debugging features, making it challenging for developers to diagnose and troubleshoot issues effectively.

In summary, considering the above-mentioned challenges encountered while integrating file storage, developing APIs and customising data structure in MongoDB Realm, the developers decided to change the API framework from MongoDB Realm to Feathers.

3.3 Final Deliverable

This sub-section dives deeper into the implementation of each component described in Section 2.

3.3.1 Application Frontend

The UX adopted tab and stack navigations. Fig. 11 illustrates a UX sitemap of the application and shows the hierarchical relationships of each component. In Fig. 11, the blue box is the navigator component, while the green box is the screen component. The root navigator comprises a few key components: a splash screen, an authentication stack navigator, a bottom

bar tab navigator, and an AR navigation screen. The splash screen serves as an intermediate indicator that shows the loading status of information from the server and local storage in the front end. The authentication and bottom bar navigators are the most complex and significant front-end components, while the AR navigation screen is the focus of this project. Tabs can be categorised into three fundamental screens: Home Page, Map Page, and Account Page, as shown in Fig. 11.

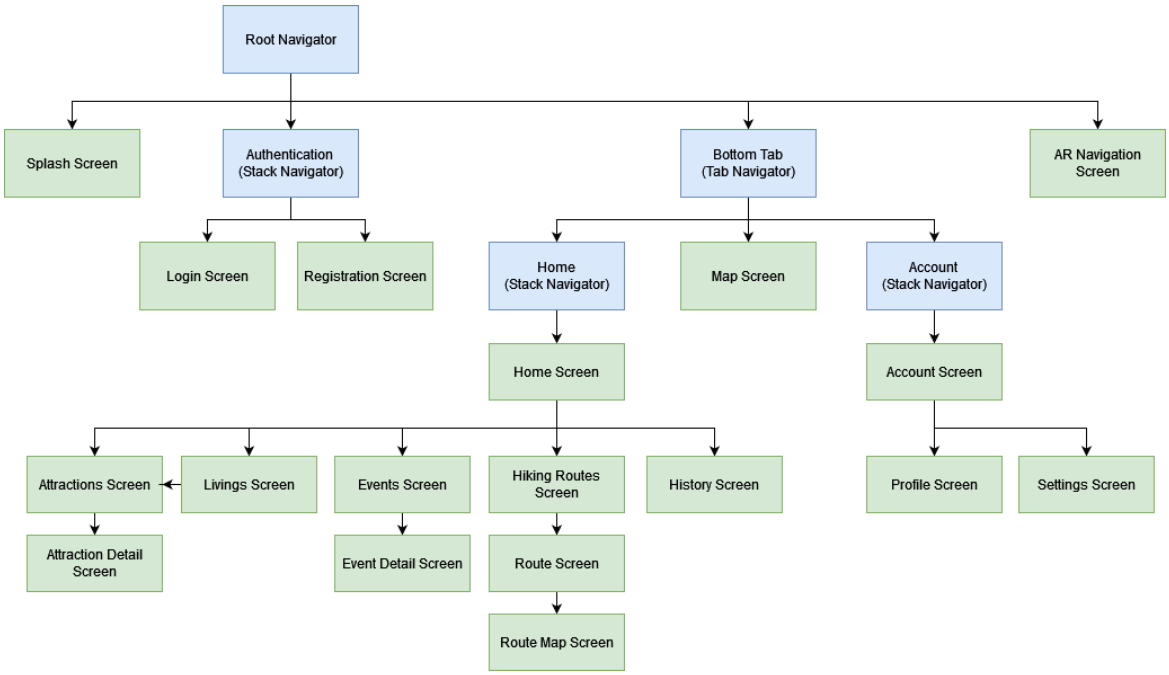


Fig. 11 UX sitemap of the front-end application

Fig. 12 displays the Home, Map, and Account pages from left to right. As can be seen, the Home Page serves as a landing page for the user to access different services in the application. The Home page consists of three sections: different categories related to Vedi River Valley, an authentication bar for anonymous users or favourite attractions for authenticated users, and a picture of the map of Vedi River Valley, as displayed from top to bottom in the leftmost frame in Fig, 12. In the top section of the Home page, different categories involving attractions, living, hiking, and events are displayed.

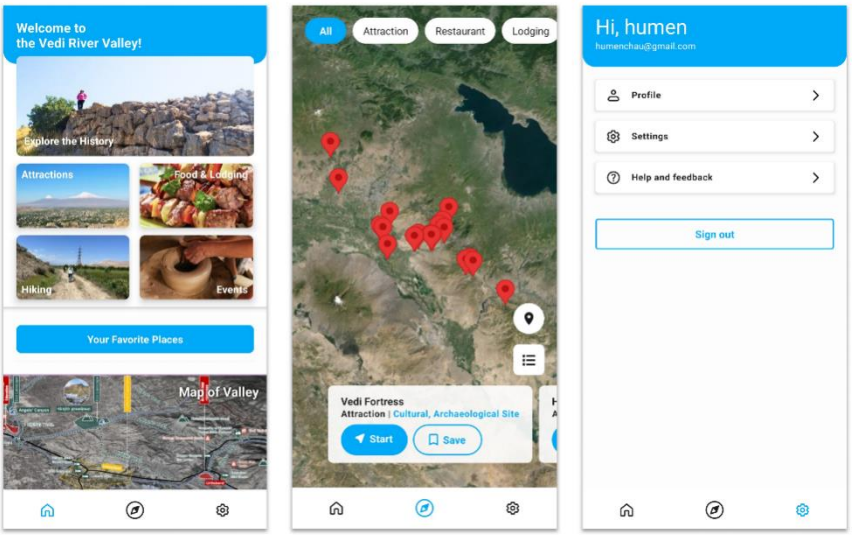


Fig. 12 Three main tab screen components

Upon launching the application, the user enters the Home page anonymously if the user is new to the application or has signed out. Fig. 13 illustrates the authentication flow and outlines the differences in screens for authenticated and anonymous users. Suppose the user is anonymous. After the “Login” button is pressed on the Home page on the leftmost in Fig. 13, the user will be directed to the Login Page to fill in the registered email and password. Before the form is submitted to the server, it will be validated by the front end and be warned with labels (Fig. 14) if there is any input error. The front end will then communicate with the back end to complete the whole authentication process, which will be further explained in subsection 3.2.3. If the users are new to the application, they can press the “Sign up” button on the Home or Login pages to access the Registration page. Apart from the mentioned screens, the Account page allows the users to access the authentication pages. As explained above in subsection 3.1.1, there is a possibility that booking and payment systems will be integrated. Hence, the sign-up form provides the contact number as an optional field for future development of notifications or subscriptions. After creating an account successfully, the user is navigated to the Login page and then is redirected to the Home page if the input email and password match the previous data during registration.

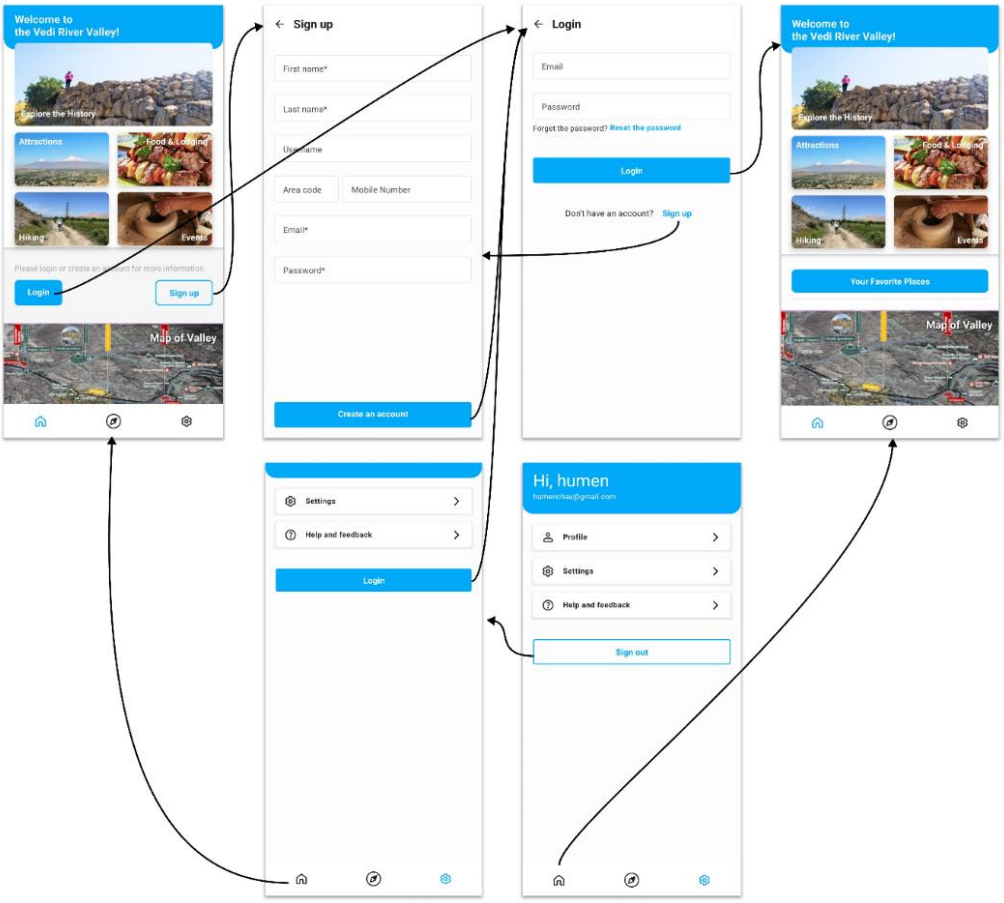


Fig. 13 The authentication flow and differences in screen components for authenticated and anonymous users

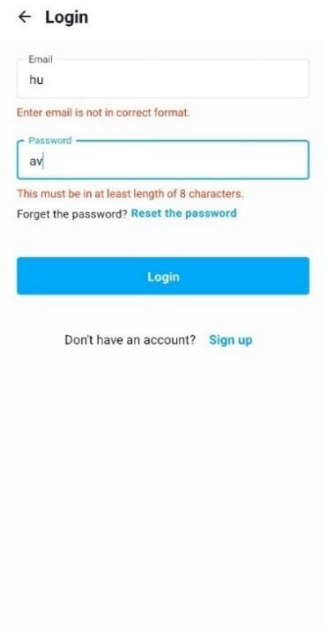


Fig. 14 Login form validation with error labels

Moving on to the attractions screens, shown in Fig. 15. Fig. 15 illustrates the Home, Attractions, Attraction Detail, and Map pages from left to right and the flow of viewing details of an attraction. The user can view a list of tourist spots after pressing the attraction panel on the home page. The page applies a paginated loading to achieve an infinite scroll on the list of tourist attractions. Initially, the page will fetch a maximum of 10 items from the server. If the user scrolls to the bottom of the list, the frond will then fetch 10 items more. On each card of the tourist attraction, name, thumbnail (if applicable), brief description, and tags are displayed. On the rightmost of the card, there will be a GPS pin icon to direct the user to the Map page with an initial position focusing on the selected tourist attraction. Aside from that, the user can also view the geographic site of the attraction on the Map page by pressing a map preview or the "View in a map" button on the Attraction Detail page, shown on the second rightmost frame in Fig. 15.

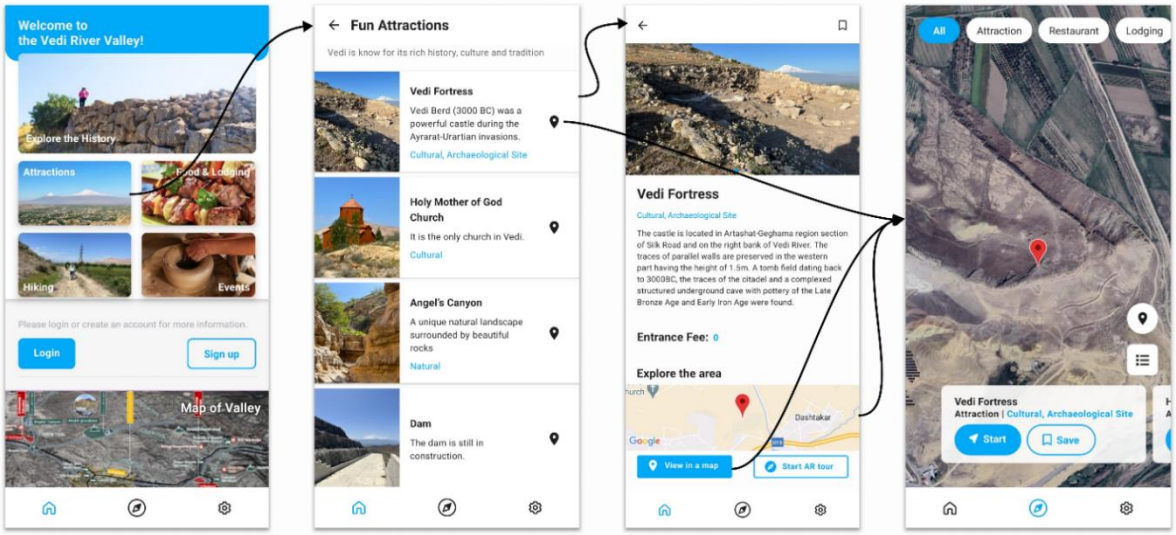


Fig. 15 The flow of viewing an attraction

The Living page is divided into two sections: restaurants and lodgings. When a user enters the screen, each section displays a maximum of ten tourist spots. A paginated list of restaurants or lodgings can be accessed by the "view all" button. Fig. 16 also shows the cards on the Living page contain additional information, such as opening hours and contact numbers. They were designed to make it easier for visitors to reach out to the spots.

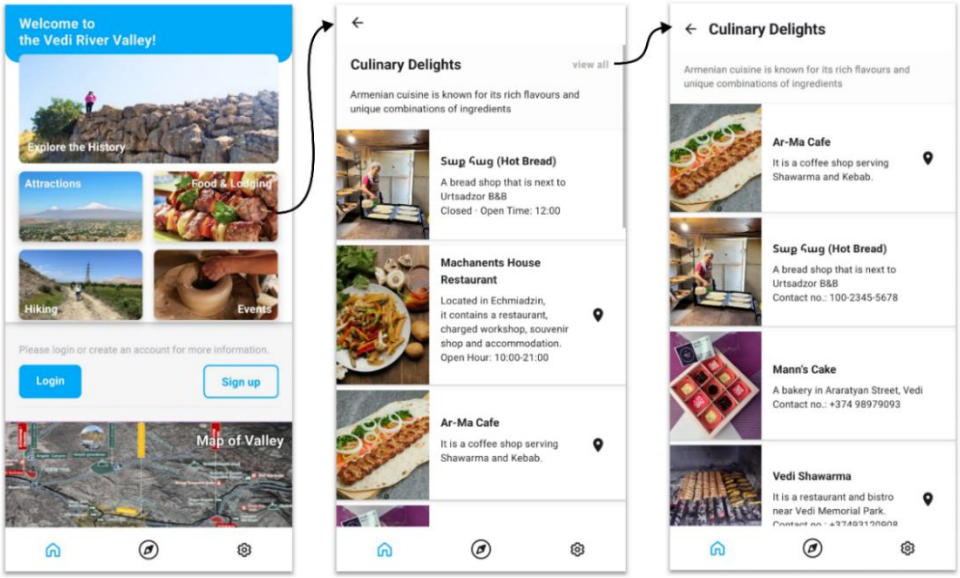


Fig. 16 The flow of viewing living, including restaurants and lodgings

Additionally, pre-designed hiking routes are available on the Routes page for users to explore recommended travel destinations. The layout and flow of the Attractions, Living, and Hiking Routes pages and details pages remain consistent, making it easy for users to navigate through them quickly. The travelling order is indicated by the yellow arrows on the map preview on the Route Detail page and the Route Map page, as seen in the two rightmost frames in Fig. 17.

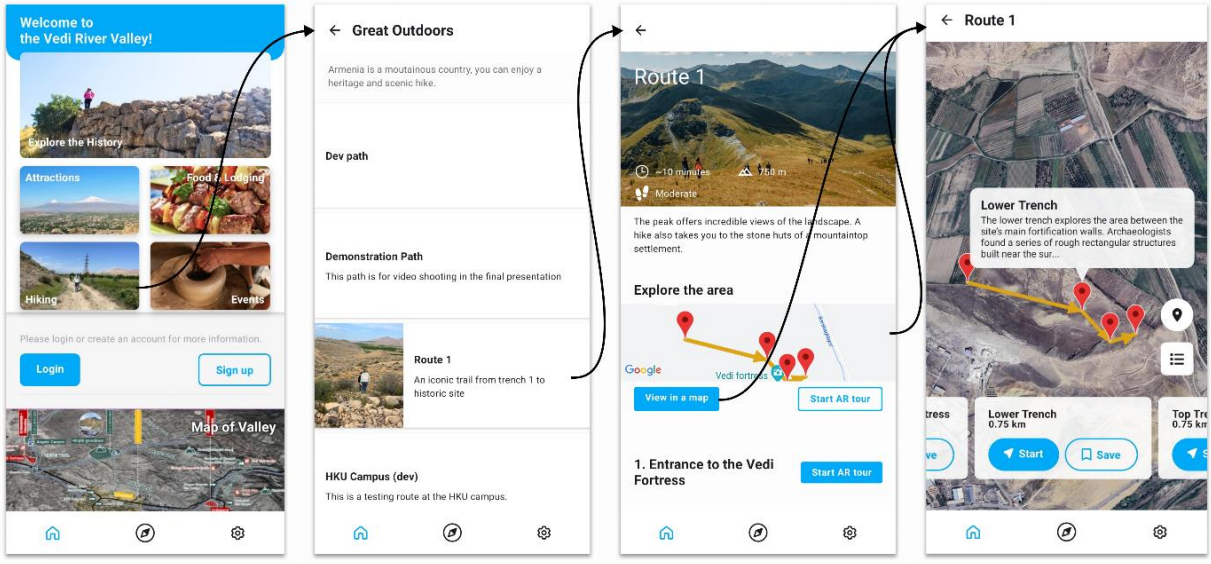


Fig. 17 The flow of viewing a hiking route

Upon pressing a pin on the maps, a callout that displays brief information is presented on the top of the pin and the bottom horizontal list scrolls to the card of the selected site automatically (the rightmost frame in Fig. 17). If the user taps on the callout, a bottom sheet

modal containing the full description and thumbnails of the selected site will appear on the screen. On the Map and Route Map pages, the users can view the list of tourist spots by pressing the list icon on the right side of the screens (Fig. 18) apart from the bottom horizontal scrolling list. Searching by the name was offered, as presented on the right side of Fig. 18. It was implemented using client code instead of server handling given that all tourist spots had already been fetched ahead in the loading of the map.

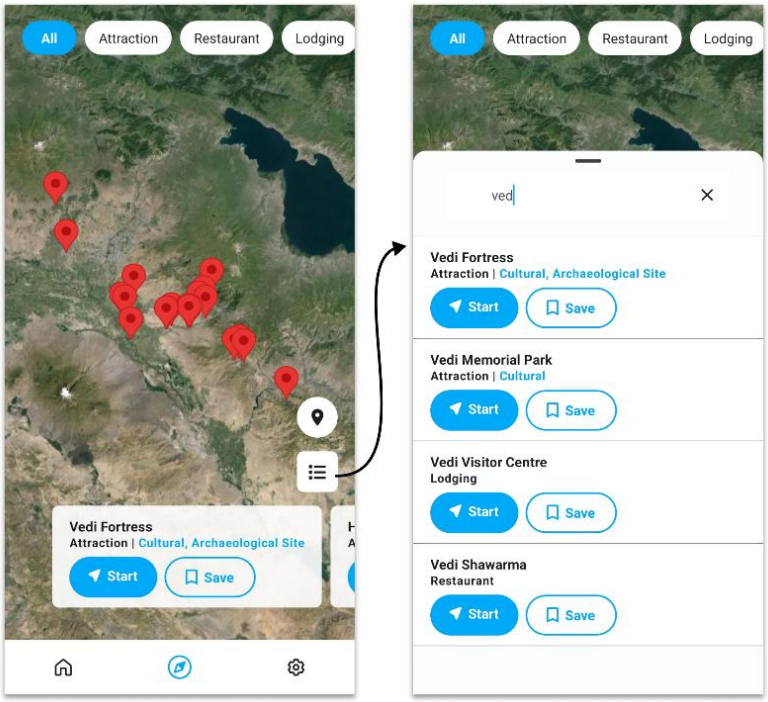
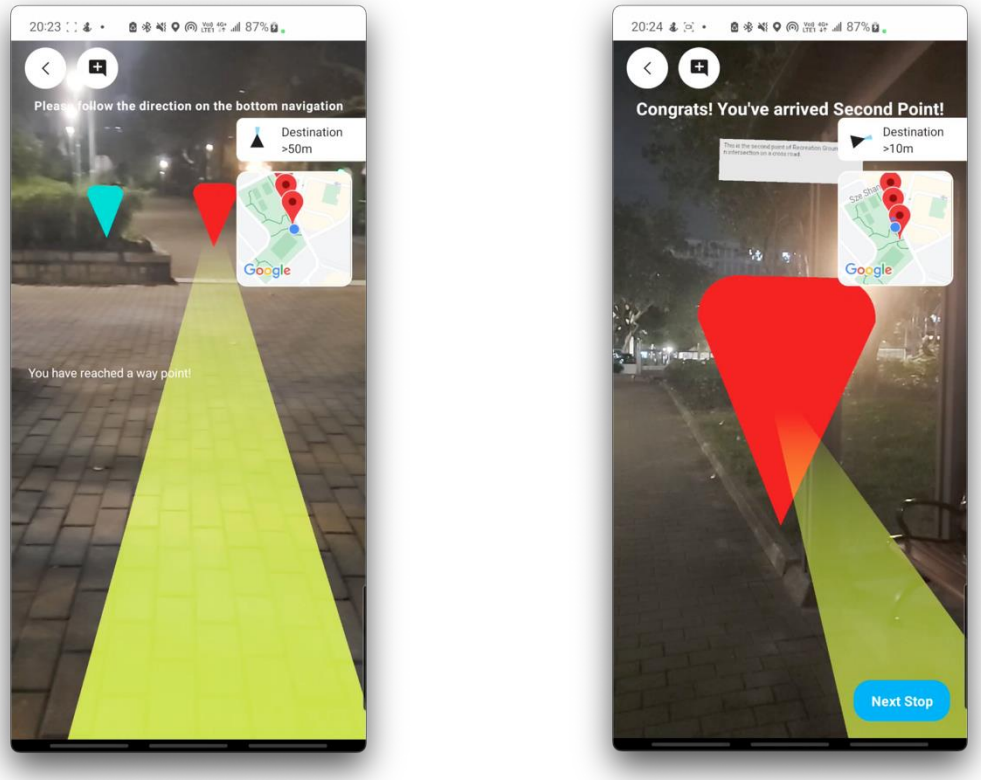


Fig. 18 The flow of viewing and searching a list of tourist spots

The users can enter the AR navigation by the “Start” buttons on the card on map pages (Fig. 17 and Fig. 18) and the “Start AR tour” buttons on the detail pages (Fig. 15 and Fig. 17). The user can experience the entire travel from the start point by the outlined button below the map preview or from a particular tourist site by the solid button aside the name on the detail page (see Fig. 17 second rightmost frame). The system will calculate the distance between the user and the POI before taking them to the AR navigation screen. If the distance exceeds five kilometres, the system will display a warning if the user attempts to use the AR function for walking.



(a) The system navigates the user to the POI *(b) The user reached the POI*

Fig. 19 AR navigation screen

The AR navigation screen consists of several UI components to suggest and lead the user to their POI. In Fig. 19(a), starting from top to bottom, the screen has an instruction line, a help notification banner for the user to keep track of the real-time distance and direction between themselves and their intended destination, and a mini map for a better illustration of the current positions of the user and the POIs. In the banner, along with the direction represented by an arrow, the degree of uncertainty of the heading angle obtained from the device's magnetometer is represented by a slice of blue section. The distance shown is an approximation, not an exact measurement with digits, to round the GPS accuracy error, which will be explained in Section 4.

The user is led by a greenish-yellow route object located beneath the user. The route object always directs to the POI. Its length depends on the distance between the user and the POI, which means it is shorter if the user is closer to the POI and vice versa. By default, the rendering position of the route object updates once the device's coordinates in AR space change. ViroReact provides frequent tracking of the camera's motion -- up to 60 frames per second. However, due to the concern of hardware performance explained later in section 4, its update frequency of rendering was changed to depend on the user's walking speed in GCS.

There is one blue and one red waypoint object in the AR space, as shown in Fig. 19(a). The blue waypoint indicates the closest point on the path relative to the user, while the red one represents the POI. A warning line in the middle of the screen (see Fig. 19(a)) suggests the user is getting closer or further to the blue waypoint. The blue waypoint and the warning line are designed to prevent the user from failing to track the path between the previously visited POI and the current POI, like the illustration demonstrated by Fig. 4(b) in subsection 2.1.2.2.

When the user reaches the POI with a radius of 20 meters, the top instruction line will note the arrival and state the name of the POI, as shown in Fig. 19(b). If the POI has a description, an information card will also pop up on the top of the waypoint and face towards the user (see Fig. 19(b)). Meanwhile, if the POI is not the last destination on the route, a “Next stop” button will appear on the bottom right corner of the screen, shown in Fig. 19(b), allowing the user to continue the trip seamlessly.

The authenticated users can express their opinions or instant feelings about the POI using the AR comment function by the top right comment icon button, shown in Fig. 20. A text field input then pops up and allows the users to fill in. Next, if the input is not empty, the user is prompted to tap on the screen to place the comment in the AR space, as demonstrated in Fig. 20. The comment is submitted and saved in the server, and thus it can appear again at the same place when the user re-enters the AR navigation screen. The comments are public to all app users. To prevent the front end from being overwhelmed by enormous amounts of rendering AR comments, the cycle of fetching and updating AR comments within 1 km of the user is set to be every hour or when the user has walked 1km.

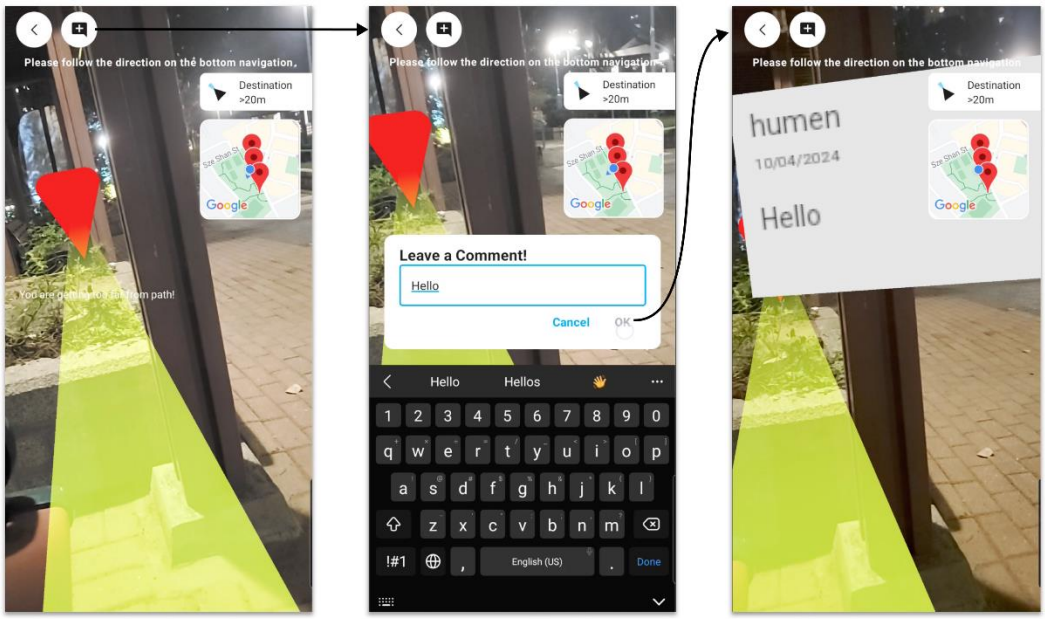


Fig. 20 The flow of AR comment

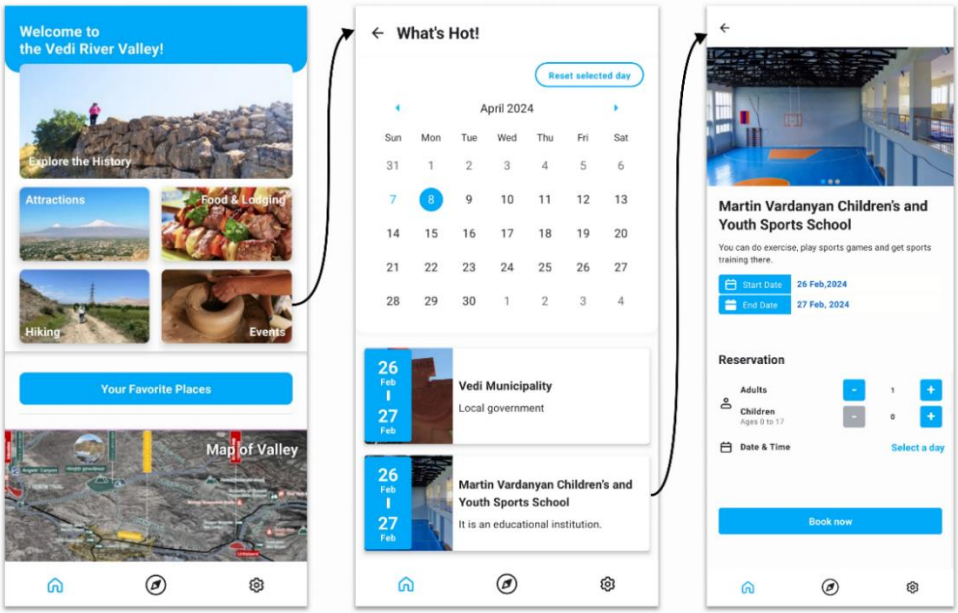


Fig. 21 The flow of viewing an event/activity

As mentioned in subsection 3.2.1, the application provides Events and Event Detail pages for personal scheduling, booking and payment. A calendar is designed to filter the local activities occurring on the picked-up date, as shown in the middle frame in Fig. 21. Below the calendar is a list of event cards displaying the name, brief description, thumbnail and dates. The user could perverse a booking on the Event Detail page. However, as the time was limited and they were out of the initial project scope, the team did not complete both the front end and back end.

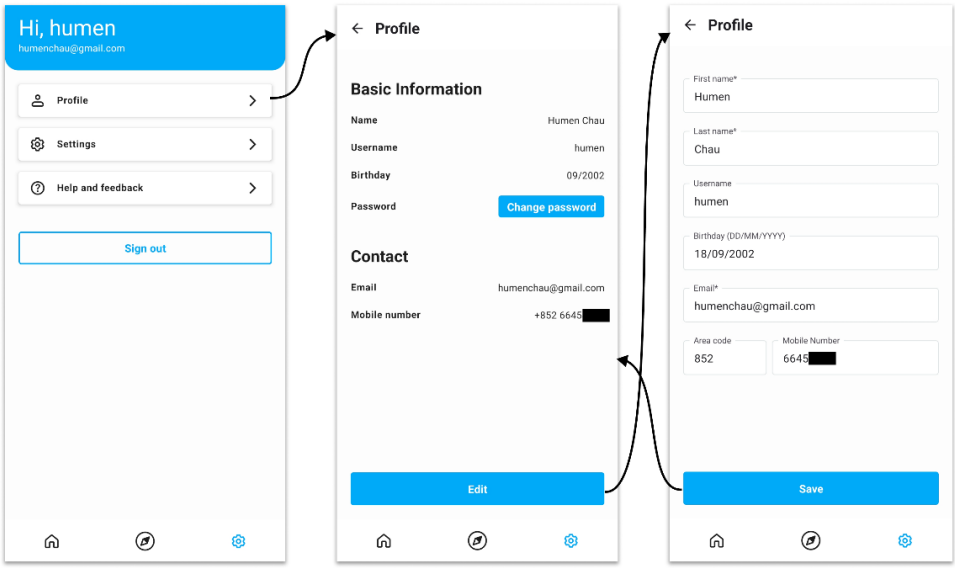


Fig. 22 The flow of editing profile

The authenticated users can edit their personal information through the Profile page. The update will first be sent to the server and then be saved in the local storage of the device. The complete illustration can be seen from Fig. 22.

The dark and light modes were not initially part of the project scope but were added as an extra feature to enhance the user experience. As a tourism application, the aim is to provide customers with the best possible personal customer experience. Unlike other applications, the application allows instant switching between light and dark modes without restarting, as Fig. 23 demonstrates the flow.

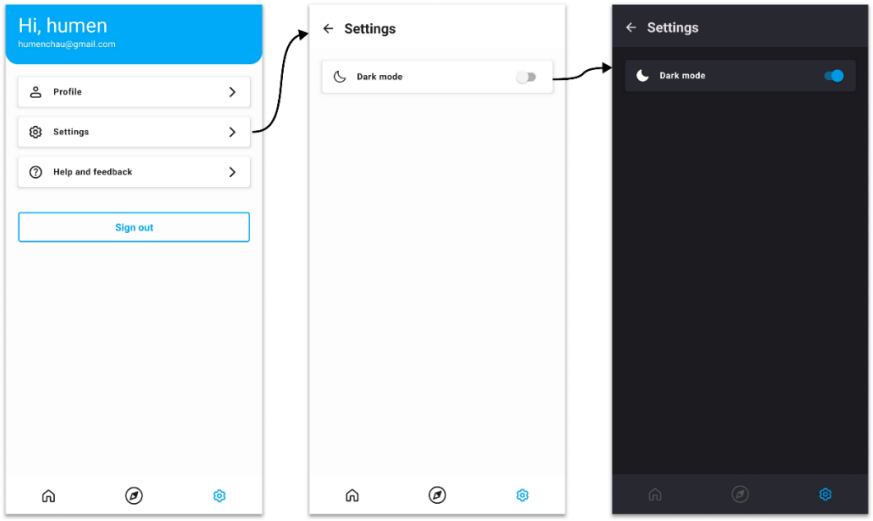


Fig. 23 The flow of switching to dark mode

The overall design of the UI is kept simple and consistent between different modes, as shown in the below Fig. 24.

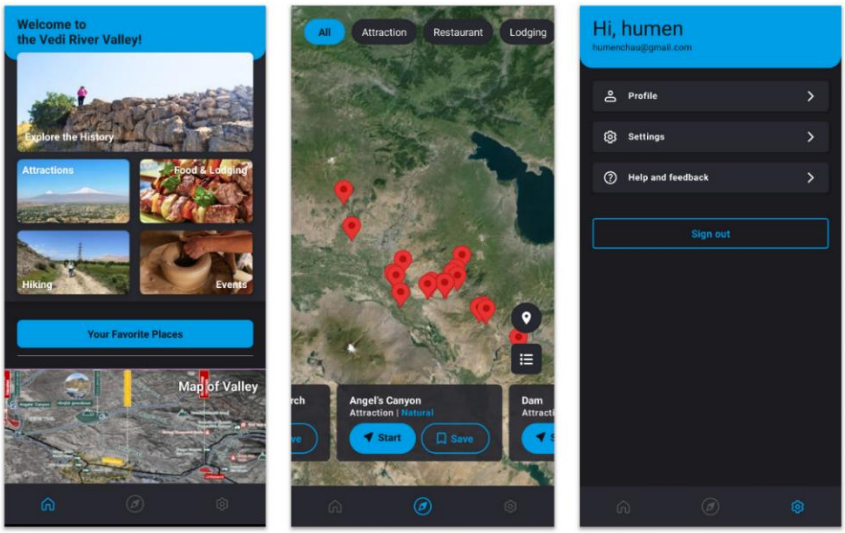


Fig. 24 An overview of the application in the dark mode

3.3.2 Admin Console Frontend

A dynamic website was built as an admin console. The admin console serves as a comprehensive data management platform for the admins to perform CRUD operations on the data collections in the database. Each page in the admin console is responsible for one data collection in the database. Since each page shares the same layout and functionality this project renders front-end components, including tables and text field inputs, based on the schema object obtained from the server. In summary, dynamic rendering and dynamic routing were applied in the admin console to optimize the scalability.

As illustrated in Fig. 25, apart from the table which uses pagination, the admin console panel comprises several components:

The navigation bar is a scrollable drawer displayed on the left side of the screen (Fig. 25(a)) and contains all data collections stored in the database. As observed from Fig. 25(a), related collections are grouped in an expandable list tile, such as `locations` and `routes` are categorized into `hike`. The grouping was specified on the server-side code.

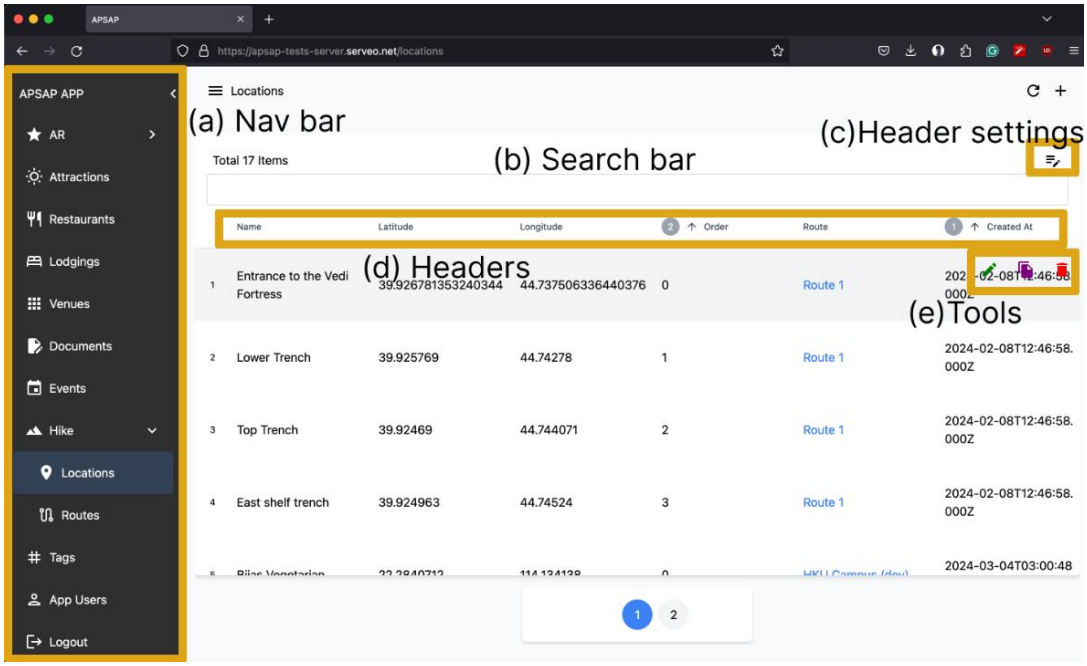
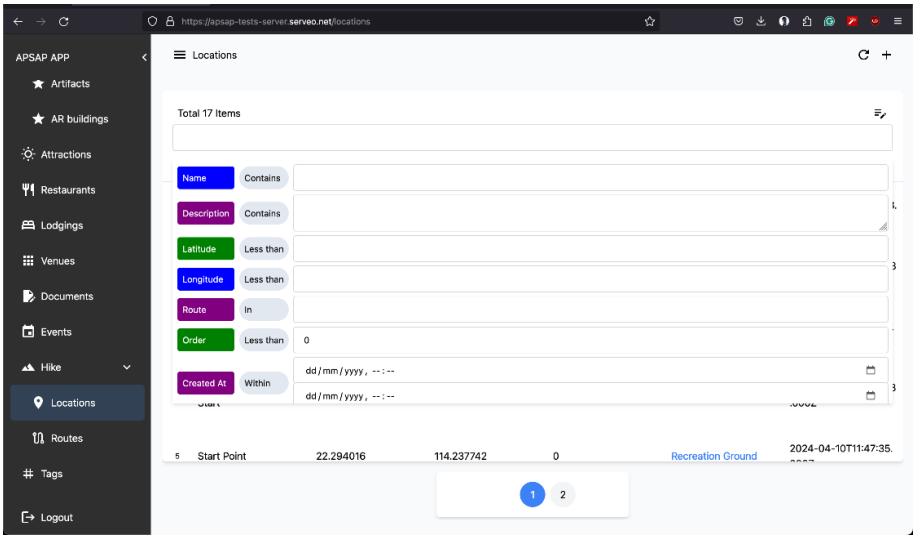
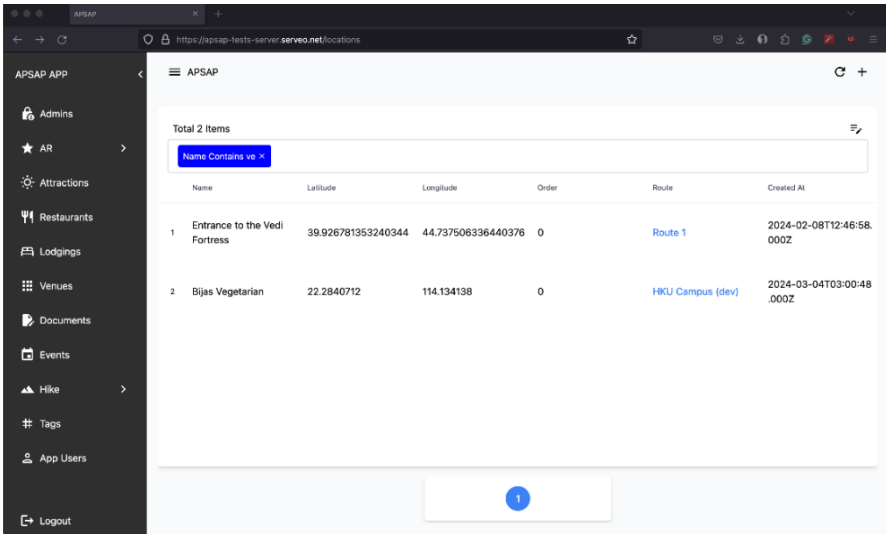


Fig. 25 Overview of the web-based admin console

The search bar on the top of the table (Fig. 25(b)) allows the admins to search data with various querying criteria related to the table columns. Fig. 26(a) highlights a floating box beneath the search bar that appears after the user clicks on the search bar. It presents all queryable attributes of the data collections and respective operators, such as contains for string attributes, and less than and greater than for number or date data fields. The user can also switch the operator of a property by clicking the grey box containing the operator (see Fig. 26(a)). After the user enters the value, the front-end table will communicate with the server to query data matching all conditions applied, as Fig. 26(b) demonstrates.



(a) A floating box beneath the search bar containing text field inputs for all queryable attributes in a table



(b) An example of querying results on location collections

Fig. 26 Querying table in the admin console through a search bar

Apart from the default display of attributes in a table as the headers, the admin can also set the table headers through the button highlighted as (c) in Fig.25. Attributes that are not embedded object type will be selectable in the header setting window, as shown in Fig. 27. The settings of each table will be saved in local storage. Therefore, each time the admin re-enters the page, the display setting of the headers can remain unchanged as the previous.

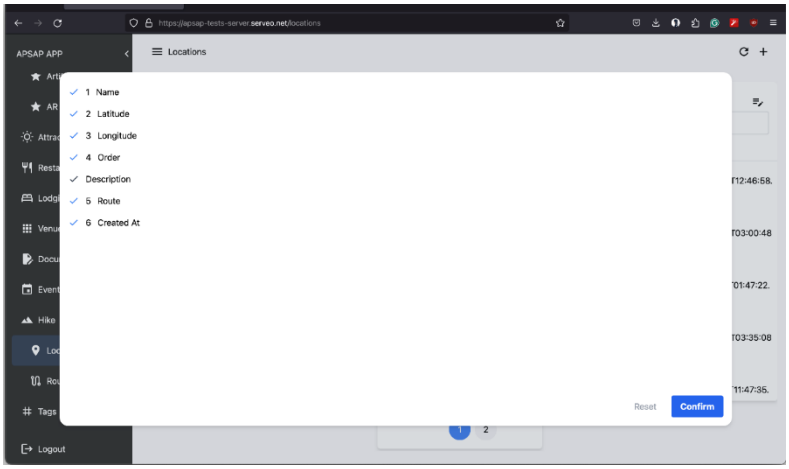


Fig. 27 The window of header setting of location collection

Meanwhile, the admin console panel offers sorting based on multiple attributes. As seen in Fig. 25(d), the `created at` and `order` have labels of one and two to indicate the order of the properties, and the upward arrows indicating both properties are sorted in ascending order. The decision of ascending or descending order based on attribute can be made by clicking the header, while the sorting orders of properties can be arranged through a long press on the

header. Like the header setting, the customisation of the sorting order of each table is cached in local storage.

Fig. 25(d) displays the editor tools available for each data. These tools, listed from left to right, are for editing, duplicating, and deleting the data. After clicking the edit button, the administrator can preview and modify the data details. This action will open an editor window containing input fields for each attribute, as illustrated in Fig. 28.

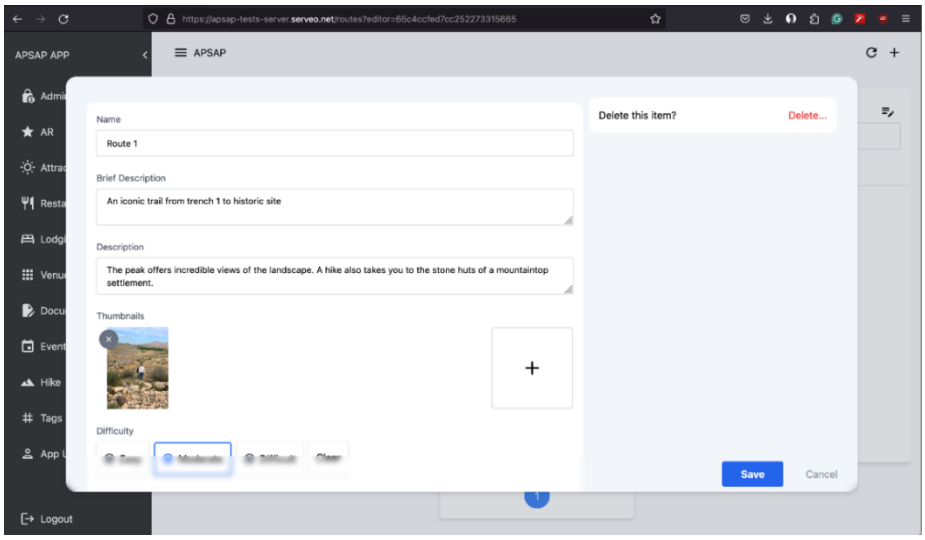


Fig. 28 The pop-up editor window that shows data details and allows admins to modify data

The duplicate button enables the admin to create a copy of data. The duplicated data retains the same values in each attribute, except the `_id` primary key, which is unique for each item in this project. The delete button is a simple feature to remove data from the database. Yet, before the deletion of an item, the system will pop up an alert and wait for the admin's confirmation to avoid accident deletion.

As mentioned in section 2.2.3, this project also provides a file storage service for saving large files such as images and 3D models. The admins can upload and manage the attachments in a media library, as demonstrated in Fig. 29. It can be accessed by a plus icon at the centre of a box in the attachment input, like the `thumbnails` field shown in Fig. 28. The types of media displayed in the media library depend on the attachment MIME types specified in the data attribute. For example, `thumbnails` allows only images, while `models` is limited to the data with MIME type of obj for model and jpg for materials. As illustrated in Fig. 29, the left side of the media library allows the admins to select preferable files, while the right side shows the details of the selected file, including name, upload date, size of the file, etc.

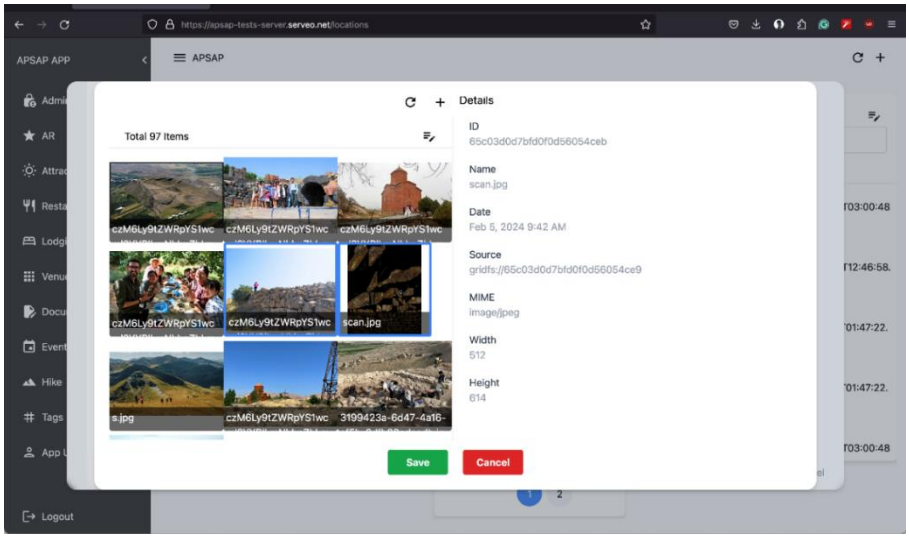


Fig. 29 The media library for displaying images stored in the database

To add a new item to the collection, the admin can click the top right add button shown in Fig. 25. The editor window (Fig. 30) will pop up and request the admin to fill in. As observed, the layout of the pop-up adding a new item window and the pop-up editor window shown in Fig. 28 is similar. In the editor window, a red-bordered text field represents a required input, while a blue-bordered text field represents the inputting field, as Fig. 30 demonstrates. In addition, data validations, such as data type, format and range checks, are implemented. An error label with details will display beneath the input if the admin enters an incorrect value (see the `latitude` field in Fig. 30). The creation and update requests with a form containing any incomplete or inaccurate values will also be blocked by the server, to prevent incorrect data from being stored or processed.

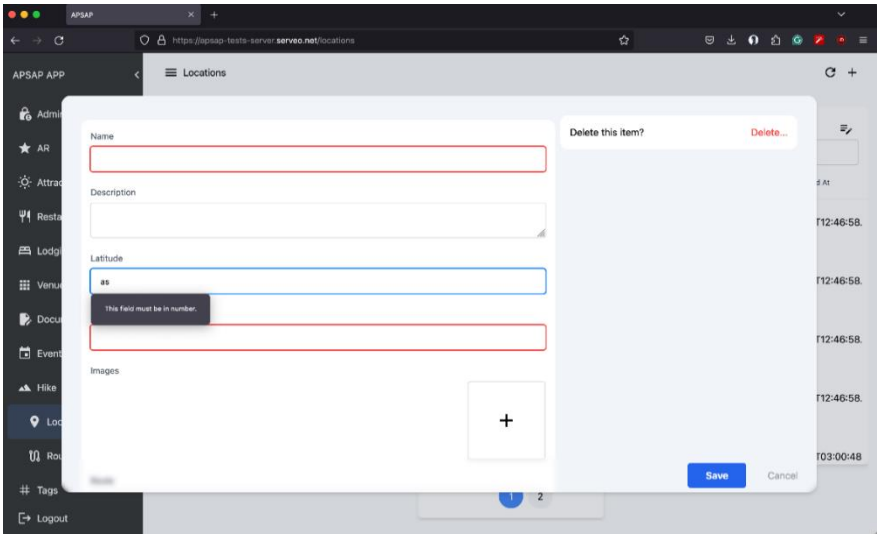


Fig. 30 The pop-up adding new item window

All data in the corresponding table will be listed for data fields with one-to-one or one-to-many relations to another table, as illustrated in Fig. 31. Links for data with one-to-one

relation (see the `route` column in Fig. 25) are also provided in the table to enable the admin to navigate through that page and edit the corresponding data quickly.

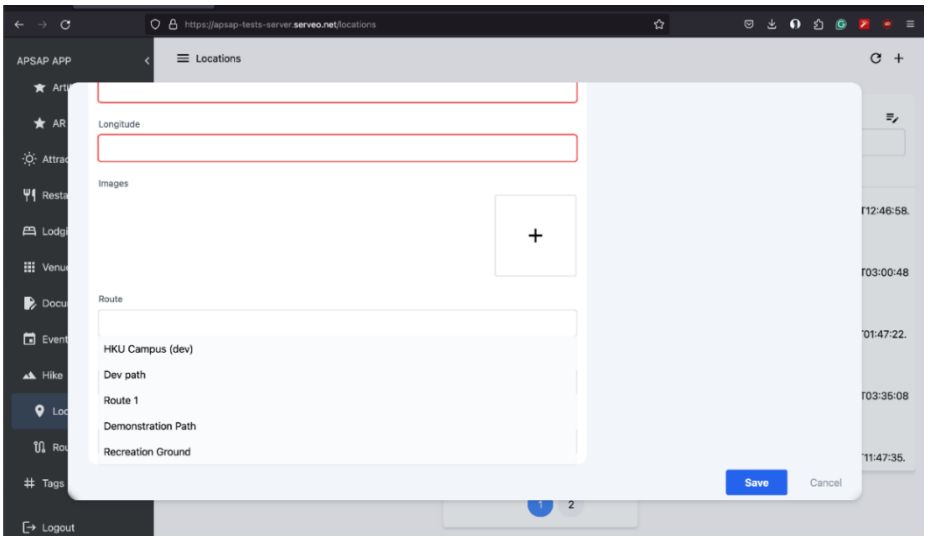


Fig. 31 The editor window with a highlight of an input field that contains one-to-one relations with another table

3.3.3 Backend

As mentioned in the Methodology section, the backend involving database and API was implemented using MongoDB and Feathers. The results of each implementation will be discussed below.

3.3.3.1 Database and API structure and setup

In this project, the server-side codes are divided into three main components, namely `api`, `db`, and `feathers`, which are also the directories shown in Fig. 32. The `api` directory is designed to contain customised services with different endpoints from the name of database collections and act as a collective service from various collections. For example, a checkout service may require calls to bookings, orders and payment from other third-party APIs. The `api` comprises two parts of the server-side codes, namely `public` and `services`, shown in the figure of the project structure. The `public` is responsible for the services provided to the mobile application, while the `services` is responsible for the services offered to the web-based admin console. The `db` directory stores data models in which role-based access controls are stated. Lastly, `feathers` serves as a package responsible for database setup, automating the conversion of data models to service object that implements CRUD action, service registration, etc, using Feathers.

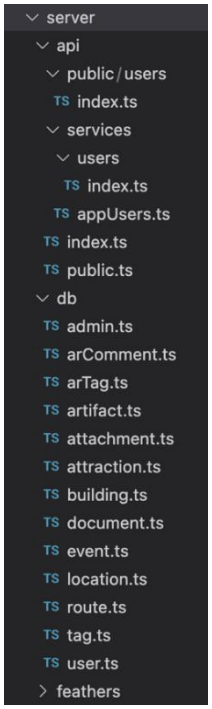


Fig. 32 Folder structure of server-side code

After MongoDB is connected, the server-side code in `feathers` acts as follows:

1. Automation of converting data models to service objects: Server-side code scans through each exported object in the `db` directory using the webpack's `requireContext` function, treats each data model as a service object and implements CRUD functionalities with access control limitations.
2. Registration of customised services: The system then procedures codes in each directory in the `api` directory.
3. Common services registration: Authentication strategies and file storage services are registered for both `public` and `service` servers.
4. Export schemas: Data models in the `db` directory will be converted into a schema object, and an endpoint of `schema` is registered for the website admin console to retrieve the schema.

Automating the conversion of data models to service objects and schema objects can be challenging for new developers to maintain codes. This approach might require time to read through documents and codes to understand the object types and functions. However, it optimizes the scalability of data structures and APIs because services can be added effectively without code repetition in writing the same CRUD functionalities and role-based access controls. Updates to data models can also be reflected instantly on the admin console, without

modifying front-end codes about the page and router. In addition, code in the `feathers` directory can be reused to perform the same function across multiple contexts. Such code reusability increases productivity in project development.

The below figure is a code snippet of the `location` data model. As can be observed, data fields such as name, latitude and longitude have the definition of type and validation syntax. The validation syntax is used by the validators of the backend and frontend, such as the example shown in Fig. 30. Apart from that, there are attributes with a dollar sign, which are hidden additions from the end users. The `$services` field indicates the endpoints, namespaces and authorisations, while the `$params.editor` field determines the front-end components and settings in the admin console panel. For example, the grouping of related collections mentioned in subsection 3.3.2 is specified by the `$params.editor.group` attribute.

```
const schema: SchemaDefExtt = {
  name: { type: String, required: true },
  desc: { type: String, $editor: { props: { multiLine: true } } },
  latitude: { type: Number, min: -90, max: 90, required: true, index: true },
  longitude: { type: Number, min: -180, max: 180, required: true, index: true },
  images: [{ type: "id", ref: "Attachment", fileType: "image" }],
  route: { type: "id", ref: "Route", required: true },
  order: { type: Number, default: 0, min: 0, required: true },

  createdAt: { type: Date, default: Date },

  $services: {
    services: {
      locations: {},
    },
    public: {
      locations: {
        hooks_Auth: ["readOnlyHooks"],
      },
    },
  },
  $params: {
    editor: {
      headers: ["name", "latitude", "longitude", "order"],
      icon: "MdLocationOn",
      group: "hike",
      groupIcon: "MdLandscape",
    },
  },
}
```

Fig. 33 Code snippet of the `location` data model

3.3.3.2 Authentication

This project implemented local and JWT authentication mechanisms. The local authentication requires the user's email and password and is used for account creation and sign-in; meanwhile, the JWT authentication authenticates the JWT in request headers and is involved in re-authentication. Re-authentication is triggered upon launching the mobile application, calling APIs and the reconnection of sockets.

Fig. 34 illustrates the login flow with a local authentication mechanism, and Fig. 35 illustrates the flow of re-authentication using a JWT. Authentication, including identification and input validation, is handled server-side, as shown in the figures. If the provided login information is correct, the server returns personal information and a JWT in both login and re-authentication. The client code will also save the returned JWT for the next re-authentication.

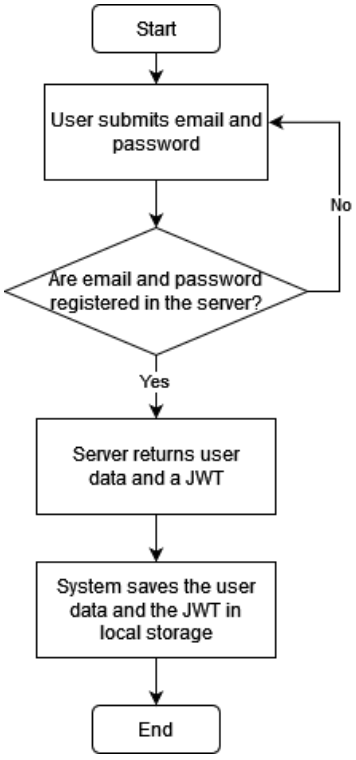


Fig. 34 The backend flow of log-in using local authentication

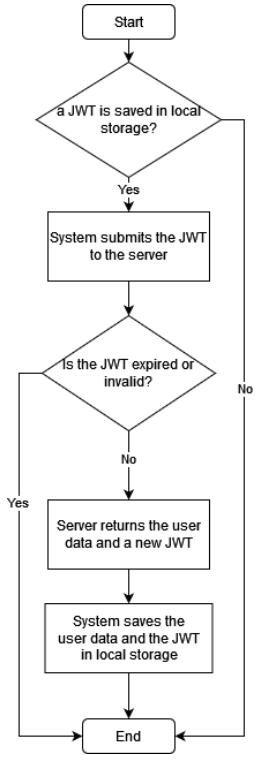


Fig. 35 The backend flow of re-authentication using a JWT

3.3.3.3 Authorisation

In the server of this project, `public` and `services` mentioned in the previous subsection 3.3.3.1 handle API calls from two front-end components, and hence the multiple types of end

users and the role-based control accesses, respectively. The details of access controls provided to various types of users are described in Table 4 below.

Table 4 Authorisation of different types of end users

	Admin Console		**Mobile App**
User types	**Admin**	**Editor**	**App User**
Access Control	Full access controls	Unable to read and edit the user's account	Read access to most data; Can edit personal account
Anonymous	No	No	Yes

The end users are split into admin and editor in the admin console. The role of the editor was developed for stakeholders mentioned in subsection 3.2.3 other than the developer and APSAP team members.

It is worth mentioning that even though the admins have complete access control, including the ability to read and write the data of app users and other admins, they cannot modify others' passwords through the admin console panel because the password field is removed in server-side return objects. Even if someone with the authenticated access token requests a change in another user's password using a POST protocol with REST API or `update` and `patch` actions with WebSocket API, that new password in the request will be deleted. A code snippet of the authorisation implementation using Feathers' hook function is demonstrated in Fig. 36 below.

```
before: {
  get: [iff(isProvider("external"), setField({ from: "params.user._id", as: "params.query._id" }))],
  create: [
    (hook: HookContext) => {
      if (!hook.params.provider) return;
      if (!hook.params.user || hook.params.user.role !== "admin") {
        throw new errors.NotAuthenticated("No permission");
      }
    },
    local.hooks.hashPassword("password"),
  ],
  patch: [
    (hook: HookContext) => {
      if (!hook.params.provider) return;
      if (!hook.params.user || (hook.params.user.role !== "admin" && !checkID(hook.params.user, hook.id))) {
        throw new errors.NotAuthenticated("No permission");
      }
      if (!hook.data.password) delete hook.data.password;
```

```
      if (hook.params.user.role !== "admin" || checkID(hook.params.user, hook.id)) delete hook.data.role;
    },
    local.hooks.hashPassword("password"),
  ],
  update: disallow("external"),
  remove: [
    (hook: HookContext) => {
      if (!hook.params.provider) return;
      if (!hook.params.user || hook.params.user.role !== "admin" || checkID(hook.params.user, hook.id)) {
        throw new errors.NotAuthenticated("No permission");
      }
    },
  ],
},
after: {
  all: [local.hooks.protect("password")],
  patch(hook: HookContext) {
    if (checkID(hook.id, hook.params.user)) _.assign(hook.params.user, hook.result);
  }
}
```

Fig. 36 Code snippet of the admin authorisation using Feathers' hook

3.3.3.4 WebSocket and REST APIs

The mobile application and the web-based admin console rely on the WebSocket API to communicate with the server. CRUD functionalities can be easily called by the service object that implements `get`, `create`, `update`, `patch`, and `remove` methods, as Table 2 in section 2.2.3 describes. WebSocket is the primary API framework for this project because it allows the registration of listeners to subscribe to the updates of collections and respond to the change and quicker data transfer compared with REST APIs. Real-time communication can be implemented using WebSocket APIs.

On the other hand, the download and preview of large files such as images and 3D models were called by the POST and GET actions of REST API.

4 Difficulties and Limitations

To date, there are several problems encountered, but some of them are expected. Most of them are related to the restrictions of development tools and devices.

4.1 Measurement Errors in GPS

The location-based navigation relies heavily on the built-in GPS and magnetometer to acquire the user's positional and directional information. While the advancements in location tracking technology have minimised the margin of error in GPS positioning to 5 meters, the settings of this application for obtaining geographical data factor in this limitation. The distance between the user and the POI is calculated using geographical coordinates, and therefore the determination of whether the user has reached the desired location is based on this computation. Suppose the user has arrived at a particular place, but the system fails to update geolocation data because its radius of uncertainty, also called the margin of error, exceeds 5 meters by a little. In that case, it will prevent the system from guiding the user to their next point of interest or displaying a notification that the user has reached their intended destination. Consequently, the system may appear unresponsive to the user. Therefore, the trade-off between real-time updates and accuracy is crucial. It is worth noting that the optimal value of 5 meters for the margin of error is already in place. However, even a small error can affect the rendering position, as the translation of global coordinates to local coordinates relative to the user’s position is involved in matching AR space with real space and updating the coordinates of AR objects.

4.2 Measurement Errors in Compass Heading Angles

Compass value can easily be distorted from ambient magnetic field noise, especially in dense city areas, and the device’s motion. Furthermore, the compass in different devices or platforms shows different stability performances. Through the experiments, Android showed unstable frequent compass value variations. Even though there were only 20 uncertainty degrees in heading value most of the time, the fluctuations and sudden jumps in values still greatly affected the performance of the directional indicator. A low-pass-filter algorithm is the current workaround to reduce the impact of noise. However, it requires users to stand still for several seconds to obtain the correct estimated result, which is not ideal for a real-time application.

4.3 Technical Requirements in Using Application

The application relies heavily on the Internet to communicate with the server, including the API calls and fetching data. However, these can be less accessible in developing countries. To minimise the payload size of each data fetching, pagination for lists is implemented. Currently, the application does not support offline mode, except the user data, which requires data caching in local storage.

On the other hand, hardware requirements are necessary for AR experience. The functions involved in the AR navigation mentioned in section 2 require vector and spatial data computations. Typically, heavy computations should be operated on the server. However, the computations in AR navigation require frequent and constant synchronisation of the user's location. In this case, an update, and a request to the server per second are necessary based on the current settings. This approach will put a heavy burden on the network communication between clients and the server and result in a performance issue in the server, especially when there are a lot of users. Juggling a heavy workload, client communications, and a limited budget for multiple servers was a challenge. Hence, computations about AR navigation are written client-side. However, the performance of the client-side computation depends on the device. Slow system response in the mobile application might result if the smartphone does not have enough processing power, memory, battery life and display quality. The developers have made an effort to reduce the requirement of computational resources, such as caching to reduce the frequency of rendering and computations and optimise the data structure.

5 Future Work

The APSAP team hopes to release a beta version of the app to conduct experiences in Vedi River Valley and usability testing in July 2024. Before the release of the application, some recommendations to overcome the limitations mentioned and improve the product are listed below:

The developers have been improving the AR navigation software and have identified several areas that need attention, particularly in terms of accuracy. To address this issue, the project team is currently exploring the integration of QR code-based image recognition, a form of marker-based AR, into the system. It is believed to enhance the accuracy of the AR navigation software and provide the app users with a more seamless experience.

During the experiments in AR navigation, it is observed that the user will easily be blocked by barriers such as buildings, vertical walls and fences when walking towards the destination in urban areas. Although the Vedi River Valley would have more open areas, leading to a better user experience in navigation, it is considered better to implement plane detection to detect obstacles and Google Maps' Road API to obtain the optimised route between waypoints and detect crossroads junctions [23]. However, as the Roads API uses a pay-as-you-go pricing model [23], it is necessary to plan and budget accordingly.

In addition, the tilt compensation algorithm to minimise the impact of measurement error in compass heading angle mentioned in section 4.2 is being researched. The tilt compensation algorithm computes the device's orientation from yaw, pitch, and roll rotations [24]. By taking the accelerometer readings into account, it is believed to enhance the estimation accuracy of compass heading angles.

To make the experience more engaging and interactive, the team is also working on enabling users to place interesting AR objects during navigation or in AR comments. This will add an element of fun and creativity to the AR navigation experience, making it more enjoyable and memorable for our users.

Front-end tasks such as bookmarks, a booking system, data selection to filter events in the calendar, and offline alerts also need attention. These enhancements will make the application more user-friendly and accessible, enabling the app users to plan and navigate their journeys more efficiently and effectively.

6 Conclusion

Various cross-media application opportunities have emerged with the advent of digital and virtual technologies. The focus of this project is to develop a cross-platform mobile application that primarily focuses on leveraging AR to revolutionize how visitors interact with archaeological sites and tourist attractions. The APSAP centres around the Vedi River Valley in Armenia, which is the main site of interest for the initiative. Through this application, the user can better understand and appreciate the cultural heritages in Armenia with an immersive experience.

This paper provided an overview of this project and discussed the techniques applied and the final deliverables. Since this project is a collaboration project with the APSAP team led by Dr Peter Cobb from the Faculty of Arts at the University of Hong Kong, the focus can be directed to the development of the mobile application and the admin console. At the same time, massive and significant data gathered from the site is provided by Dr Peter Cobb and his team. The data includes information about the history of the site, their respective locations and descriptions, and the cultural significance of the site. We have been working closely with Dr Cobb's team to ensure that the data is accurate and up to date.

While there have been challenges during the development, such as GPS inaccuracies, magnetometer instabilities, and necessary adjustments to UX and backend design, these obstacles had been overcome thanks to the project team's unwavering determination, collaboration with Dr Cobb's team and the valuable guidance of the supervisor, Dr Choi. There are still some areas of improvement to be implemented in the future, including machine learning or algorithms to enhance the estimation and calibration of compass headings, integration of marker-less AR, and more. Nevertheless, it is believed that the current product shows the core functionalities of location-based AR and can provide a comprehensive and informative platform about the Vedi River Valley as a tourism and AR map application.

References

[1] "HKU archaeological team excavates at one of the major fortress-settlements in the Armenian Highlands – the highest part of the ancient Near East," The University of Hong Kong, 9 9 2019. [Online]. Available: https://www.hku.hk/press/news_detail_19858.html. [Accessed 18 10 2023].

[2] M. Onuoha, "Side-by-side images expose a glitch in Google's maps," QUARTZ, 6 6 2017. [Online]. Available: https://qz.com/982709/google-maps-is-making-entire-communities-invisible-the-consequences-are-worrying. [Accessed 5 4 2024].

[3] "Download the official mobile app for Hong Kong Disneyland," Hong Kong Disneyland, [Online]. Available: https://www.hongkongdisneyland.com/mobile-app/. [Accessed 5 4 2024].

[4] "Universal Studios Japan," Universal Studios Japan, [Online]. Available: https://www.usj.co.jp/web/en/us/enjoy/app. [Accessed 5 4 2024].

[5] Frank van Diggelen, Per Enge, "The World's first GPS MOOC and Worldwide Laboratory using Smartphones," 18 9 2015. [Online]. Available: https://www.ion.org/publications/abstract.cfm?articleID=13079. [Accessed 6 4 2024].

[6] "AR and VR for archaeology," [Online]. Available: https://www.carraro-lab.com/ar-vr-archaeology/.

[7] "What Are The Different Types of Augmented Reality," Nextech3D.ai, 25 5 2022. [Online]. Available: https://www.nextechar.com/blog/what-are-the-different-types-of-augmented-reality. [Accessed 22 10 2023].

[8] "Configuring Plan Detection for AR Foundation," Unity, 29 12 2020. [Online]. Available: https://learn.unity.com/tutorial/configuring-plane-detection-for-ar-foundation#. [Accessed 14 9 2023].

[9] "Tracking and Anchors," 2022. [Online]. Available: https://viro-community.readme.io/docs/tracking-and-anchors. [Accessed 15 1 2024].

[10] “Simple Lowsspass Filter,” [Online]. Available: https://dobrian.github.io/cmp/topics/filters/lowpassfilter.html. [Accessed 24 4 2024].

[11] “Expo,” Expo, 2023. [Online]. Available: https://expo.dev/.. [Accessed 12 9 2023].

[12] “EAS Build,” Expo, [Online]. Available: https://docs.expo.dev/build/introduction/. [Accessed 7 4 2024].

[13] “Flutter,” Google, [Online]. Available: https://flutter.dev/. [Accessed 7 4 2024].

[14] V. Kariatukaran, “What Should You Choose from Flutter vs. React Native in 2024?,” RADIX, 21 3 2024. [Online]. Available: https://radixweb.com/blog/flutter-vs-react-native. [Accessed 7 4 2024].

[15] “viro,” 2020. [Online]. Available: https://github.com/viromedia/viro. [Accessed 15 1 2024].

[16] “Augmented Reality in React-Native,” 10 8 2020. [Online]. Available: https://stackoverflow.com/questions/63341035/augmented-reality-in-react-native. [Accessed 7 4 2024].

[17] “Future of ViroReact,” 3 5 2020. [Online]. Available: https://github.com/viromedia/viro/issues/853. [Accessed 7 4 2024].

[18] “MongoDB vs Firebase: Which Is The Best Database in 2022,” MQoS Technologies, 1 1 2022. [Online]. Available: https://medium.com/mqos-technologies/mongodb-vs-firebase- which-is-the-best-database-in-2022-aff873566586. [Accessed 16 9 2023].

[19] R. Sharma, “Firebase vs MongDB: Difference Between Firebase & MongoDB,” upGrad, 18 6 2023. [Online]. Available: https://www.upgrad.com/blog/firebase-vs-mongodb/. [Accessed 16 9 2023].

[20] J. Richman, “Firebase vs. MongoDB: Major Differences,” Estuary, 3 5 2023. [Online]. Available: https://estuary.dev/firebase-vs-mongodb/. [Accessed 16 9 2023].

[21] “BSON Types,” MongoDB, [Online]. Available: https://www.mongodb.com/docs/manual/reference/bson-types/. [Accessed 23 10 2023].

[22] “feathers,” feathers, [Online]. Available: https://feathersjs.com/. [Accessed 8 4 2024].

[23] “Roads API overview,” Google Maps Platform, 17 4 2024. [Online]. Available: https://developers.google.com/maps/documentation/roads/overview. [Accessed 18 4 2024].

[24] T. Ozyacilar, “Implementing a Tilt-compensated eCompass using Accelerometer and Magnetometer Sensor,” 11 2015. [Online]. Available: https://www.mikrocontroller.net/attachment/292888/AN4248.pdf. [Accessed 25 4 2024].

[25] “Augmented Reality in the Archaeological Field: Redefining the Past,” Lets Nurture, 25 9 2021. [Online]. Available: https://www.letsnurture.ca/blog/technology/augmented-reality-in-the-archaeological-field-redefining-the-past/. [Accessed 22 10 2023].

[26] “Simulator User Guide,” Apple Developer, 15 2 2018. [Online]. Available: https://developer.apple.com/library/archive/documentation/IDEs/Conceptual/iOS_Simulator_Guide/Introduction/Introduction.html. [Accessed 22 10 2023].

[27] “AR and VR for archaeology,” CarraroLAB, 19 4 2023. [Online]. Available: https://www.carraro-lab.com/ar-vr-archaeology/. [Accessed 22 10 2023].

[28] “Find & improve your location’s accuracy,” Google Maps, [Online]. Available: https://support.google.com/maps/answer/2839911?hl=en&co=GENIE.Platform%3DAndroid. [Accessed 6 4 2024].