



Faculty of computers and information Technology

Site Seeing Navigation Tool

Antiques Museum Application

Supervised By:

DR. Yasser Abdelhamid

Eng. Mohamed Khedr

Team Members:

Rodina Adel Mahmoud	20-00394
Mariam Hamza Helmy	20-00614
Adham Ashraf Elhoussieny	20-01126
Mohamed Ali	20-01233
Ramez Amgad Farouk	20-01248
Cherif Adel	20-00794
Abdallah Hisham	20-00309

Acknowledgements

First, we'd like to thank DR. Yasser Abdelhamid, who helped us learn a lot about this project. His ideas and comments aided in the completion of this project. his unwavering support, superb direction, and inspiring words throughout the course of our senior project. We would also like to thank all our professors for their support and advice throughout our education, as well as their assistants, particularly Eng. Mohamed Khedr for all the guidance he provided to us

Abstract:

This project addresses the problem of indoor navigation within large museum spaces. The issue poses challenges for visitors who often struggle to locate specific exhibits or facilities.

To tackle this problem, we employed a combination of interactive mapping techniques. Our approach involved developing a mobile application with a user-friendly interface that guides visitors through the museum. The app includes features such as pathfinding, exhibit information, and multi-language support.

Our findings indicate that the application significantly improves visitor navigation and enhances the overall museum experience. This study contributes to the field of indoor navigation systems by providing a scalable solution that can be adapted for various large indoor environments.

Contents

Acknowledgements	3
Abstract:.....	4
CHAPTER 1:INTRODUCTION	
1.1 Introduction:	8
1.2 Problem Statement:	8
1.3 Problem Solution:.....	9
1.4 Significance of the Project	10
1.5 Project aim and objectives:	11
1.6 Project Scope:	12
1.7 Documentation Overview:.....	12
CHAPTER 2: RELATED EXISTING SYSTEMS	
2.1: introduction.....	13
2.2: Existing Systems	13
2.3: Overall Problems of Existing Systems	14
2.4: Overall Solution Approach	15
2.5: Summary	16
CHAPTER 3: SYSTEM REQUIREMENTS ENGINEERING AND PLANNING	
3.1: Introduction	17
3.2: Feasibility Study.....	17
3.3: Targeted Users	19
3.4: Functional Requirements.....	22
3.5: Non-Functional Requirements.....	22
CHAPTER 4: SYSTEM DESIGN	
4.1: Introduction	24
4.2: Data Flow Diagram (DFD)	24
4.3: UML Use Case Diagram	25
4.4: UML Sequence Diagram	26
4.5: Graphical User Interface Design (GUI)	27

4.6: Datasets	28
4.7: Tools, Technologies and Techniques	32
CHAPTER 5: SYSTEM IMPLEMENTATION	
5.1: Introduction	33
5.2: Database Implementation	33
5.3: Chatbot Implementation	39
5.4: GUI Implementation	39
CHAPTER 6: SYSTEM TESTING	
6.2: Packages Explanation	60
Packages	60
6.3: Code Explanation	62
1.Main.dart	62
2.Languages_Screen.dart	65
3.Home_Screen.dart	72
4.Antique_screen.dart	73
5.Collection_Screen.dart	79
6.Gallery_Screen.dart	82
7.map_Screen.dart	86
8.Splash_Screen.dart	91
6.4: System Testing	93
6.5: Summary	103
Chapter 7: PROJECT CONCLUSION AND FUTURE WORK	
7.1: Introduction	104
7.2: Overall Weaknesses	104
7.3: Overall Strengths	105
7.4: Future Work	106
7.5: Summary	109

List of Tables/Figures

Figure 1: Existing System.....	13
Figure 2:Data Flow Diagram	24
Figure 4 : Use case Diagram	25
Figure 4: Sequence Diagram.....	26
Figure 5: Graphical User Interface (GUI)	27
Figure 6: FlutterFigure	28
Figure 7: Splash screen	47
Figure 8: Language ScreenFigure	48
Figure 9: Home ScreenFigure	49
Figure 10: Deutch Home ScreenFigure	50
Figure 11: Spanish Home screenFigure	51
Figure 12: Arabic Home screenFigure	52
Figure 13: Collection ScreenFigure	53
Figure 14: Antique Description ScreenFigure	54
Figure 15: Chatbot ScreenFigure	55
Figure 16: MenuFigure	56
Figure 17: GalleryFigure	57
Figure 18: SearchFigure	58
Figure 19: Map screenFigure	59

CHAPTER 1: INTRODUCTION

1.1 Introduction:

The Antiquities Museum Indoor Mapping App aims to enhance the visitor experience by providing a cutting-edge navigation tool within the museum premises. This app is designed to address the challenges faced by tourists in efficiently exploring the vast and historically rich exhibits of the Antiquities Museum.

1.2 Problem Statement:

Tourists often encounter difficulties navigating the extensive halls and exhibits of the Museum. Traditional maps, while useful, may not be sufficient for a seamless and enjoyable experience. These static maps lack real-time updates and interactive features, making it challenging for visitors to locate specific exhibits, amenities, or their current position within the museum. Additionally, the complexity of large museums with multiple floors and interconnected sections can lead to confusion and frustration.

In such environments, visitors may find themselves repeatedly consulting their maps, asking for directions, or inadvertently missing key attractions. This can detract from the overall experience, especially for those with limited time. Moreover, traditional maps do not cater to the diverse needs of visitors, such as providing accessible routes for individuals with mobility impairments or offering information in multiple languages.

Implementing such a solution not only improves visitor satisfaction but also aligns with the Museum's goal of making art and culture accessible and engaging for all.

1.3 Problem Solution:

Our project addresses the need for an intuitive indoor mapping app to enhance visitor navigation and engagement, and a chatbot for providing information

The App enabling visitors to effortlessly locate themselves within the museum and find their way to desired exhibits and amenities. The app supports multiple languages, catering to the diverse demographic of museum visitors, and includes accessibility options to assist those with mobility impairments.

Additionally, the app integrates a chatbot that serves as a virtual assistant for museum visitors. The chatbot can provide immediate answers to common questions, offer detailed information about exhibits, this not only reduces the need for visitors to seek out museum staff for help but also enhances their engagement by providing a more interactive and informative experience.

By combining an interactive indoor map with an intelligent chatbot, our project ensures that visitors have all the tools they need to navigate the museum efficiently and enjoyably. This innovative solution not only improves visitor satisfaction but also helps the museum achieve its goal of making art and culture more accessible and engaging for everyone.

1.4 Significance of the Project

The significance of this project lies in its potential to dramatically enhance the visitor experience at the museum. By providing an intuitive indoor mapping app combined with an intelligent chatbot, the project addresses several key issues faced by museum Visitors:

1. **Enhanced Navigation:** The indoor mapping app allows visitors to navigate the museum with ease, reducing the frustration often associated with getting lost or struggling to find specific exhibits. Real-time GPS tracking ensures that visitors always know their exact location and can quickly find the shortest path to their desired destinations.
2. **Improved Accessibility:** The app supports multiple languages and includes features designed for visitors with mobility impairments, making the museum more accessible to a broader audience. This aligns with the museum's mission to be inclusive and welcoming to all visitors.
3. **Increased Engagement:** The chatbot offers an interactive way for visitors to learn about exhibits, providing detailed information and answering questions on-demand. This level of engagement helps to deepen the visitor's connection with the exhibits, fostering a more educational and enriching experience.
4. **Efficient Resource Management:** By providing visitors with a self-service tool for navigation and information, the app reduces the demand on museum staff, allowing them to focus on more complex inquiries and tasks. This

improves overall operational efficiency and ensures that staff can provide higher quality assistance when needed.

5. **Visitor Satisfaction:** A smoother, more enjoyable visit increases overall satisfaction, encouraging repeat visits and positive word-of-mouth. This can lead to higher attendance rates and increased revenue for the museum.

1.5 Project aim and objectives:

The primary aim of this project is to develop an innovative indoor mapping and navigation system, coupled with a responsive chatbot, to significantly enhance the visitor experience at the museum. The system will provide intuitive, multilingual support, and interactive information access, ensuring that visitors can explore the museum's exhibits effortlessly and gain insightful knowledge throughout their visit.

1-To create an efficient and user-friendly indoor mapping app for the antiques museum

2-Develop an interactive and visually appealing map of the museum.

3.Implement location services to accurately track and guide users within the museum.

4.Ensure compatibility with both iOS and Android platforms.

5.Integrate features for exhibit information, restroom locations, and other amenities.

6.Enhance user engagement through additional educational content and multimedia features.

1.6 Project Scope:

The project encompasses the development and implementation of an advanced indoor navigation system and an interactive chatbot for the museum. The scope covers several crucial aspects to ensure a successful and impactful deployment. This includes digitizing the museum's floor plan and exhibits, and developing user-friendly interfaces for seamless navigation, and the app will support multiple languages, allowing users to select their preferred language for maps and navigation instructions.

1.7 Documentation Overview:

Chapter 1: Presenting Related Existing Systems, the problems that this system have and the solutions

Chapter 2: Discussing different literature reviews about existing projects then concluding the best fit to this project, presenting System Analysis and Design which examines the collected data then showing system's deployment methodology.

Chapter 3: System requirements engineering and planning the requirements of the system

Chapter 4: provides a comprehensive overview of the system design, covering both structured and object-oriented methodologies, The chapter concludes with a summary, encapsulating the key elements of the system design to provide a solid foundation for development and implementation.

Chapter 5: Discussing how to implement the system, integration between components and testing for it.

Chapter 6: Presenting the testing scenarios of the system which ensures that all implemented processes are working correctly.

Chapter 7: Presenting conclusion of what is done and the future work of the system.

CHAPTER TWO: RELATED EXISTING SYSTEMS

2.1: introduction

The aim is to contextualize our project within the current technological landscape and highlight the gaps our solution seeks to fill. Existing systems for indoor navigation often utilize technologies, Museum tour guide applications typically provide information about exhibits and may include features such as audio guides, virtual tours, and interactive maps. By examining these systems, we can identify the common challenges faced by users and assess how well these solutions meet the needs of museum visitors. This analysis will help us understand the shortcomings of current systems and inform the design and development of our own project, ensuring it addresses these gaps and provides a more comprehensive and user-friendly solution for enhancing the museum experience.

2.2: Existing Systems

Louvre Museum Application

Explore galleries of louvre with guided tours , and maps

Inside the app:

1-Map

2-Room navigation

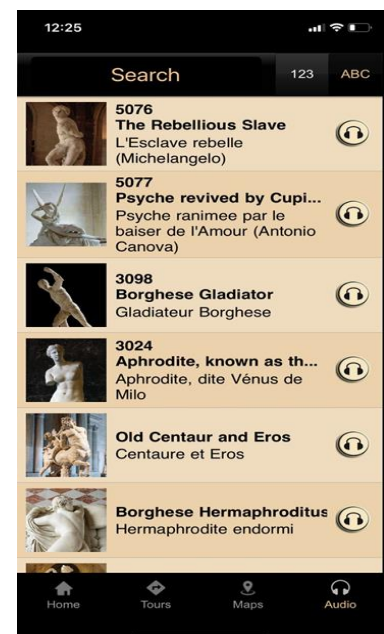
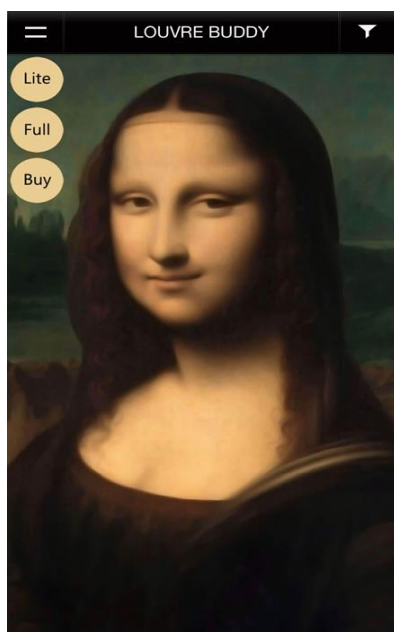


Figure 1: Existing System

2.3: Overall Problems of Existing Systems

1. **Limited Functionality:** Existing indoor navigation and museum tour guide systems often have limited functionality, offering basic features such as mapping and exhibit information. They may lack advanced capabilities such as real-time updates, personalized recommendations, or interactive experiences, limiting their usefulness to users seeking a more comprehensive and engaging museum experience.
2. **Mobile Responsiveness:** Many existing systems may not be optimized for mobile devices, leading to issues with responsiveness and usability on smartphones and tablets. This lack of mobile optimization can result in a subpar user experience, as visitors may encounter difficulties accessing information or navigating the app while exploring the museum.
3. **Cluttered Layout:** Some indoor navigation apps and museum tour guides suffer from a cluttered layout, with an overwhelming amount of information presented in a disorganized manner. This can confuse users and make it challenging for them to find the information they need quickly and efficiently, detracting from the overall user experience.
4. **Navigation Issues:** Despite their intended purpose of assisting users with navigation, existing systems may still encounter navigation issues such as inaccuracies in location tracking, routing errors, or difficulties in identifying points of interest. These navigation issues can frustrate users and lead to a loss of trust in the app's reliability, ultimately hindering their ability to explore the museum effectively.

5. **Unpleasant Color Coordination:** The visual design of some existing systems may lack aesthetic appeal or effective color coordination, resulting in an unappealing or even jarring user interface. Poor color choices can make it difficult for users to distinguish between different elements of the app, impacting readability and usability. Additionally, unpleasant color schemes may detract from the overall user experience and diminish the app's visual appeal.

2.4: Overall Solution Approach

The overall solutions implemented in our indoor mapping and museum tour guide app aim to address the identified shortcomings of existing systems, ensuring a more seamless and enjoyable experience for museum visitors. Here's an overview of the solutions we've incorporated:

1. **Mobile Optimization:** We've prioritized mobile optimization, ensuring that our app is fully responsive and user-friendly across different devices and screen sizes. Through responsive design principles, we've optimized layouts, navigation menus, and content presentation to provide a consistent and intuitive experience for users accessing the app on smartphones or tablets.
2. **Streamlined User Interface:** Our app boasts a clean and intuitive user interface with a streamlined layout that presents information in a clear and organized manner. By minimizing clutter and prioritizing essential content, we've simplified navigation and enhanced usability, allowing visitors to find relevant information and navigate the app effortlessly.

3. **Aesthetic Design:** Our app's visual design emphasizes aesthetic appeal, with carefully chosen color schemes, typography, and graphical elements. By prioritizing visual aesthetics, we've created an engaging and immersive environment that captivates users and enhances their overall enjoyment of the museum visit.

2.5: Summary

We examined the limitations of existing indoor mapping and museum tour guide systems, highlighting issues such as limited functionality, mobile responsiveness challenges, cluttered layouts, navigation issues, and unpleasant color coordination. To address these shortcomings, we devised comprehensive solutions, including enhancing functionality, optimizing for mobile devices, streamlining the user interface, and prioritizing aesthetic design. These solutions aim to provide museum visitors with a seamless, engaging, and memorable experience, ultimately improving their overall satisfaction and enjoyment during their visit.

CHAPTER THREE: SYSTEM REQUIREMENTS ENGINEERING AND PLANNING

3.1: Introduction

In chapter, we dive into the systematic process of defining, analyzing, and documenting the requirements for the development of an app for the Antiquities Museum in Bibliotheca Alexandrina. This phase is crucial as it lays the foundation for the entire project, ensuring that the app meets the needs and expectations of its users while aligning with the museum's objectives.

3.2: Feasibility Study

Before proceeding with the development of the app, it is imperative to conduct a feasibility study to assess the viability and potential success of the project. This study will examine various aspects, including technical, economic, operational, and scheduling feasibility. By conducting a comprehensive feasibility analysis, we can determine whether the project is worth pursuing and identify any potential challenges or constraints that need to be addressed by using the following steps:

- 1- **Define the Objectives:** Clearly outline the objectives of the feasibility study. Identify what you aim to achieve and what questions need to be answered regarding the project's feasibility.
- 2- **Gather Information:** Collect all relevant information about the project, including the museum's goals and objectives, available resources (financial,

human, and technological), existing systems and infrastructure, and potential risks and constraints.

3- Technical Feasibility:

- Assess the technical feasibility of developing the app by evaluating the availability of required technologies, expertise, and infrastructure.
- Determine if the museum has the necessary hardware and software resources to support app development and maintenance.
- Evaluate the compatibility of the app with existing systems and platforms used by the museum.

4- Operational Feasibility:

- Evaluate the operational feasibility of implementing and maintaining the app within the museum's environment.
- Consider factors such as staffing requirements, training needs, workflow changes, and potential disruptions to museum operations.
- Assess the willingness and ability of museum staff to adopt and support the app.

5- Schedule Feasibility:

- Assess the schedule feasibility by estimating the time required to develop, test, and deploy the app.
- Consider any time constraints or deadlines that need to be met
- Identify potential risks and dependencies that may impact the project timeline.

6- Risk Assessment:

- Identify and analyze potential risks and challenges associated with the project, such as technical issues, and time
- Develop strategies to mitigate or address these risks to ensure the project's success.

7- **Document Findings and Recommendations:**

- Compile the findings of the feasibility study into a comprehensive report, documenting the analysis, conclusions, and recommendations.

8- **Decision Making:**

- Based on the findings of the feasibility study, make an informed decision about whether to proceed with the development of the app, modify the project scope, or abandon the project altogether.

3.3: Targeted Users

1. **Museum Visitors:**

Individuals who visit the Antiquities Museum in Bibliotheca Alexandrina to explore its exhibits, artifacts, and collections.

Characteristics: Diverse demographic backgrounds, varying levels of familiarity with museum environments, interests in archaeology, history, and cultural heritage.

Needs and Expectations: Easy access to information about exhibits, navigation assistance within the museum, engaging and immersive experiences, opportunities for learning and discovery.

2. **Researchers:**

Scholars, academics, and researchers who visit the museum for academic purposes, such as conducting research, studying artifacts, and accessing scholarly resources.

Characteristics: Specialized knowledge and expertise in archaeology, history, and related disciplines, specific research interests and objectives.

Needs and Expectations: Access to comprehensive and authoritative information about museum collections, research tools and resources, support for data analysis and documentation, opportunities for collaboration and networking with other researchers.

3. **Students:**

Students from educational institutions, including schools, colleges, and universities, who visit the museum as part of their curriculum or academic studies.

Characteristics: Varied ages and academic levels, diverse educational backgrounds and interests, varying levels of prior knowledge about archaeology and history.

Needs and Expectations: Access to educational resources and materials related to museum exhibits, interactive learning experiences, support for academic projects and assignments, opportunities for guided tours and educational programs

4. **Educators:**

Teachers, educators, and instructors who bring groups of students to the museum for educational purposes or incorporate museum visits into their curriculum.

Characteristics: Experience in teaching and education, familiarity with curriculum standards and educational objectives, interest in using museums as learning resources.

Needs and Expectations: Access to educational materials and resources aligned with curriculum standards, support for designing and implementing museum-based learning activities, tools for assessment and evaluation of student learning outcomes.

5. **General Public:**

Members of the public who may not have a specific academic or research interest but visit the museum out of curiosity or for leisure.

Characteristics: Varied demographic profiles, diverse interests and motivations for visiting the museum, ranging from cultural enrichment to leisure and entertainment.

Needs and Expectations: Access to engaging and informative content about museum exhibits, user-friendly navigation features, opportunities for interactive and immersive experiences, convenient access to practical information such as opening hours, ticket prices, and visitor guidelines

3.4: Functional Requirements

1. **Map Rendering:** The system should be able to render an interactive map of the museum, displaying exhibits, halls, amenities, and other relevant locations.
2. **Artifact Information:** The system should provide detailed information about artifacts, exhibits, artwork, and other points of interest within the museum, including descriptions, historical context, and multimedia content where available.
3. **Language Switching:** Users should have the ability to change the language of the app interface and content to suit their preferences, enhancing accessibility for international visitors.
4. **Chatbot Integration:** Integration of a chatbot feature allows users to interact with the system to ask questions, request assistance, or receive additional information about exhibits and museum facilities in a conversational manner.

3.5: Non-Functional Requirements

1. **Performance:** The system should deliver fast response times and smooth interaction, ensuring minimal latency when rendering maps, providing navigation guidance, and retrieving information about exhibits. Response times for user queries and interactions should be within acceptable limits to maintain a seamless user experience.

2. **Scalability:** The system should be scalable to accommodate a growing number of users and exhibit without significant degradation in performance. It should be able to handle peak loads during periods of high visitor traffic, ensuring reliability and responsiveness even under heavy usage.
3. **Usability:** The system should be intuitive and user-friendly, with a well-designed interface that enables easy navigation, search functionality, and access to information. User interactions should be straightforward, requiring minimal learning curve for visitors of all ages and technical backgrounds.
4. **Compatibility:** The system should be compatible with a wide range of devices and platforms, including smartphones, tablets, and web browsers, to ensure accessibility for all visitors. It should be responsive and adaptable to different screen sizes, resolutions, and operating systems, providing a consistent experience across various devices and platforms.

CHAPTER FOUR: SYSTEM DESIGN

4.1: Introduction

In this chapter, we delve into the comprehensive design framework of our indoor mapping and museum tour guide system. The chapter is structured to cover both the structured and object-oriented design methodologies utilized to create a robust and user-friendly system

We present the Graphical User Interface (GUI) design, emphasizing user experience and interaction design. This chapter aims to provide a thorough understanding of the system's architecture and design considerations, paving the way for its implementation and deployment.

4.2: Data Flow Diagram (DFD)

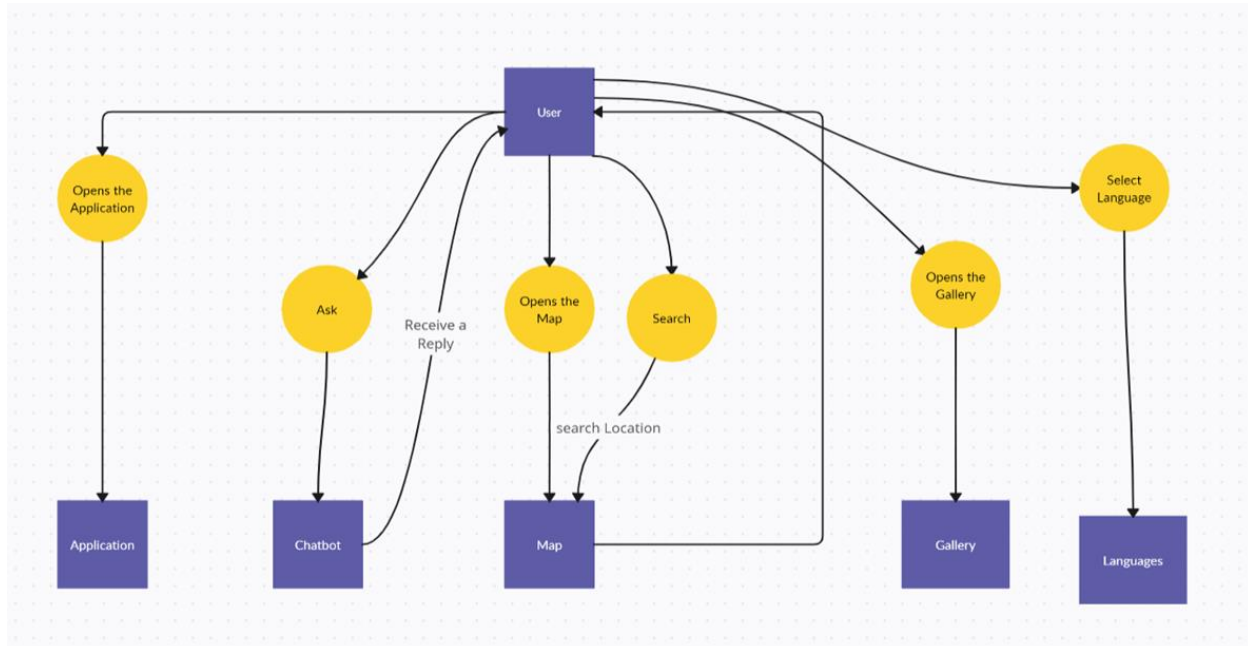


Figure 2 : Data flow Diagram

This DFD shows the different paths the user can take in the application. Starting with the welcome page, the User can ask the chatbot about any

information and the chatbot will respond, open the map and can choose specific location and it will show the name of the location, open the gallery so he will see images of the antiques there, and can select language.

4.3: UML Use Case Diagram

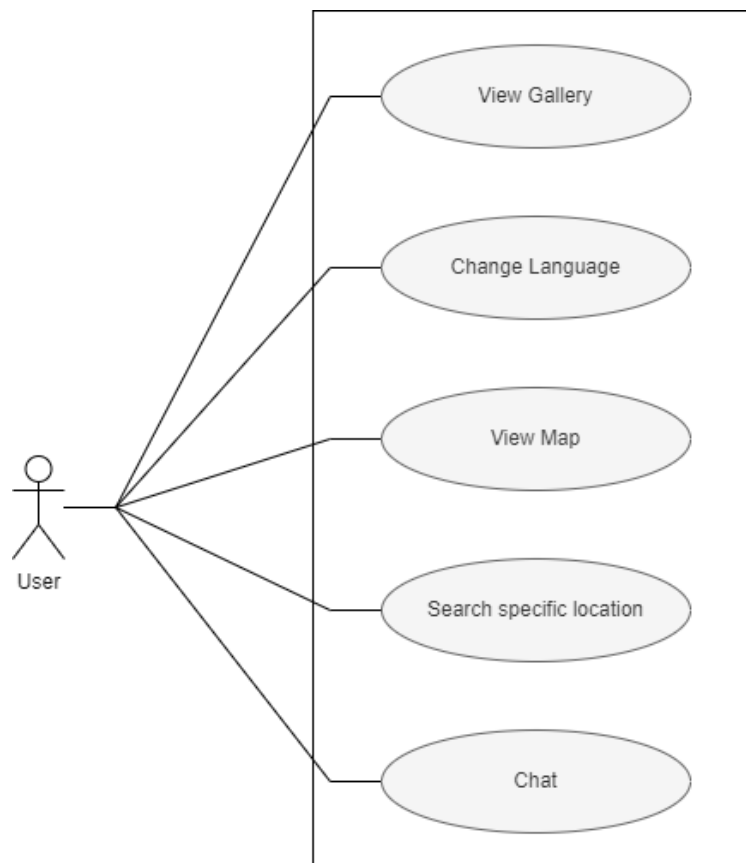


Figure 3: Use Case Diagram

User interact with the system by selecting the type of data view gallery, change language, view map , search specific location , chat with chatbot

4.4: UML Sequence Diagram

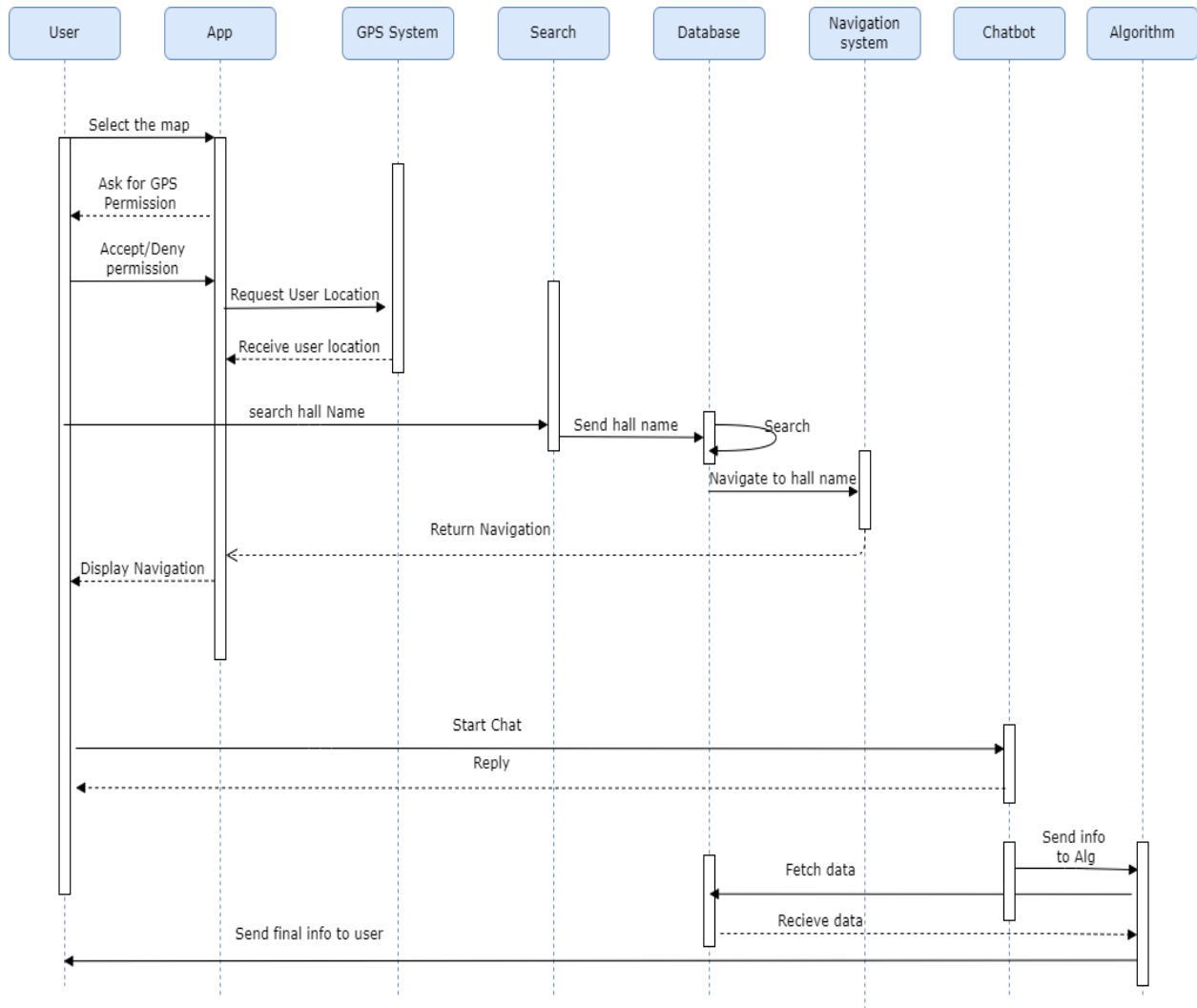


Figure 4: Sequence Diagram

User Interacts with the system by accepting or denying location, search about hall name and start navigation

4.5: Graphical User Interface Design (GUI)



Figure 5: Graphical User Interface (GUI)

4.6: Datasets

FRAMEWOK: FLUTTER

What is Flutter and Why Flutter is Top Choice to Build Apps?

Key Takeaways:

- Flutter is a mobile app development framework by Google that enables
- developers to build high-performance, visually appealing, and natively
- compiled apps for multiple platforms from a single codebase.
- The key advantages of using Flutter include customizable widgets, cross
- platform capabilities, a fast development cycle, and strong community
- support, because of which it is a popular choice for app development.
- Flutter has hot reload feature, rich pre-built tools and widgets, and cross
- platform capabilities help developers create responsive user interfaces, iterate
- quickly, and deploy apps on multiple platforms effortlessly.

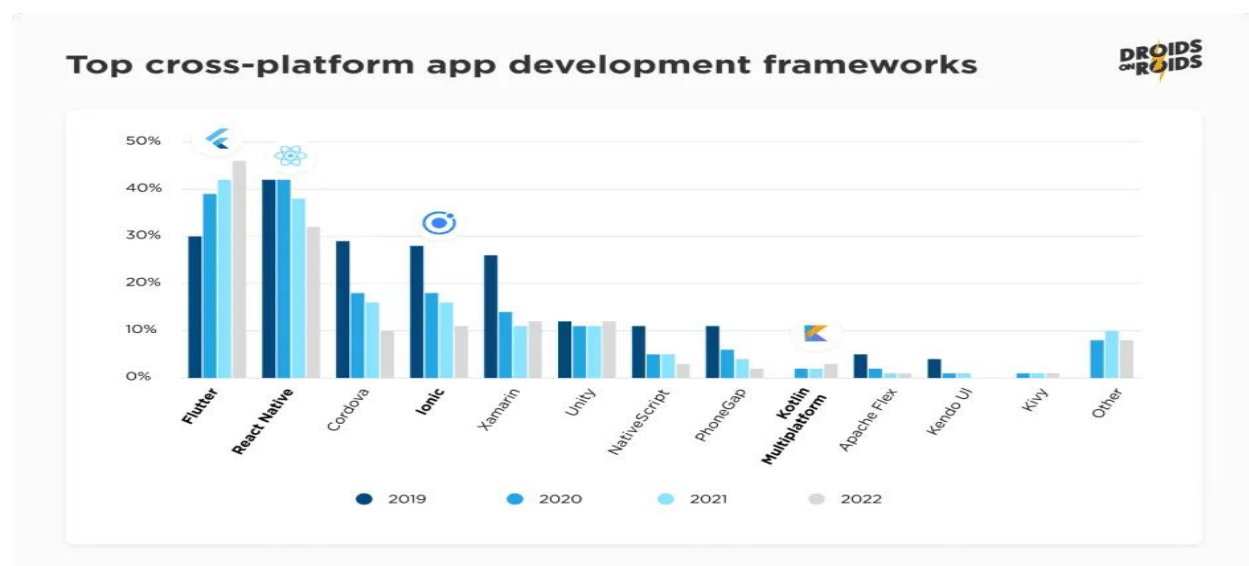


Figure 6: Flutter

1- What is Flutter?

Flutter is a framework developed by Google to create web apps, mobile apps, and embedded apps using a single codebase. Flutter mobile apps perform well and can compete with native mobile apps. The first release of Flutter came in 2017, and the Flutter project was started under the name of Sky.

What was the purpose of creating the Flutter framework?

The purpose of making the Flutter framework is to enable developers to create native-like performing mobile apps on multiple platforms with a single codebase.

The Flutter framework includes two important parts:

- **A Software Development Kit (Flutter SDK):** The mobile SDK is a collection of

software development tools that help developers to write a mobile app code. The tools available in Flutter are APIs for unit and integration tests, Dart DevTools, and Flutter & Dart command-line tools.

- **Flutter Widgets** (A Library-based on Widgets): Widgets are a collection of reusable UI elements (text buttons, text inputs, sliders, scrolling, and styling). It allows developers to build an application's UI depending on the client's needs. Widgets help to design layout, handle user interaction, and UIViews of the app.

You can check a short brief on the history of the Flutter mobile app framework in the following image.

2- Which programming language is used in the Flutter framework?

Flutter framework is built using a programming language called Dart.

3- How Does Flutter Work?

Flutter apps use the Dart language, and it allows to compile the app code ahead of time into a native machine code (iOS/Android). Then, the rendering engine (built with C++) converts app code into Android's NDK and iOS's LLVM. Both the pieces

of code are rendered into the wrapper Android and iOS platforms and results into .apk or .ipa files for final output.

4- Why Should You Use Flutter for App Development?

To give you reasons why you should use the Flutter framework, we have divided this section into two parts.

1. Some statistics on the Flutter framework
2. Features of Flutter framework

5- Features of the Flutter app development framework

Flutter helps developers to develop fast and scalable apps. Following are the Features:

- Flutter widgets
- AOT code compilation
- Hot reload

- Single codebase

6- What are the Advantages of Flutter for Android Developers?

- Writing Code Faster
- Single Codebase for Multiple Platforms
- Architecture of Flutter
- GUI and Widgets Components
- Testing is Easy and Fast

FAQ About Flutter Framework

1.Which are the top apps built using the Flutter framework?

- Google Ads
- Google Pay
- ByteDance
- eBay
- Nubank

2.Is it possible to create web applications or desktop apps using Flutter?

Yes, it is possible to create web applications and desktop apps using Flutter.

The single code gets compiled and also works on the web.

3.Which is the best framework between Flutter and React Native for cross platform mobile apps?

Both frameworks are best to build mobile apps. Flutter is famous because of its easy-to-use feature while React Native is famous for developing personalized UI of mobile apps.

4.Is Flutter better than Java?

Flutter is an SDK for creating cross-platform applications while Java only supports to development of Android apps.

5. Is Flutter easy to learn?

Flutter is easy to learn compared to Swift, Java programming language or React Native library. Because the learning curve of this framework is small compared to core language for Android or iOS app programming language.

4.7: Tools, Technologies and Techniques

- Firebase database software - used to store data in a database that contains information about rooms and antiques description
- Figma (UI) for designing the app's user interface, buttons, and layout.
- Sketch (UX) to improve the user experience

CHAPTER FIVE: SYSTEM IMPLEMENTATION

5.1: Introduction

This chapter introduces the project components' implementation processes and how they interact with one another using various techniques and tools.

5.2: Database Implementation

Firebase

What is firebase

Firebase is a set of backend cloud computing services and application development platforms provided by Google. It hosts databases, services, authentication, and integration for a variety of applications, including Android, iOS, JavaScript, Node.js, Java, Unity, PHP, and C++.

Why use firebase

Advantages:

1. Reliable & Extensive Databases
 - Realtime Database
 - *Cloud Firestore*
2. Fast & Safe Hosting
3. Google Analytics
4. Firebase Cloud Messaging for Cross-Platform
5. Free Multi-Platform Firebase Authentication
6. Firebase Testing Services to Improve App Quality.

7.Increment in Revenues with App Indexing API

8.Free Use of Firebase Dynamic Links

9.Machine Learning Capabilities

Database Storage

Antiquities-Museum ▾


Storage

gs://antiquities-museum.appspot.com

Upload file

<input type="checkbox"/>	Name	Size	Type	Last modified
<input type="checkbox"/>	Images/	—	Folder	—
<input type="checkbox"/>	Group 447.png	34.68 KB	image/png	Jun 3, 2024
<input type="checkbox"/>	baner_1-min.jpg	357 KB	image/jpeg	Jun 3, 2024
<input type="checkbox"/>	baner_1.jpg	7.29 MB	image/jpeg	Jun 3, 2024

Group 447.png



Name

Group 447.png

Size

35,517 bytes

Type

image/png

Created

Jun 3, 2024, 10:47:59 PM

Updated

Jun 3, 2024, 10:47:59 PM

File location

▾

Other metadata

▾

Antiquities-Museum ▾

Storage

Files

Rules

Usage

Extensions

Protect your Storage resources from abuse, such as billing fraud or phishing

Configure App Check

×

gs://antiquities-museum.appspot.com > images

Upload file

<input type="checkbox"/>	Name	Size	Type	Last modified
<input type="checkbox"/>	antiques/	—	Folder	—
<input type="checkbox"/>	collections_icons/	—	Folder	—

Antiquities-Museum ▾ Storage

gs://antiquities-museum.appspot.com > images > collections_icon.

Upload file

<input type="checkbox"/>	Name	Size	Type	Last modified
<input type="checkbox"/>	1.png	7.38 KB	image/png	Jun 4, 2024
<input type="checkbox"/>	2.png	34.68 KB	image/png	Jun 4, 2024
<input type="checkbox"/>	3.png	16.07 KB	image/png	Jun 4, 2024
<input type="checkbox"/>	4.png	7.52 KB	image/png	Jun 4, 2024
<input type="checkbox"/>	5.png	10.63 KB	image/png	Jun 4, 2024
<input type="checkbox"/>	6.png	7.55 KB	image/png	Jun 4, 2024
<input type="checkbox"/>	7.png	14.23 KB	image/png	Jun 4, 2024
<input type="checkbox"/>	8.png	8.92 KB	image/png	Jun 4, 2024

3.png

Name
[3.png](#)

Size
16,459 bytes

Type
image/png

Created
Jun 4, 2024, 12:38:46 PM

Updated
Jun 4, 2024, 12:38:46 PM

File location

Antiquities-Museum ▾

Storage

Files Rules Usage Extensions

Protect your Storage resources from abuse, such as billing fraud or phishing [Configure App Check](#)

gs://antiquities-museum.appspot.com > images > antiques

Upload file

<input type="checkbox"/>	Name	Size	Type	Last modified
<input type="checkbox"/>	Alabaster Offering Table.jpg	142.66 KB	image/jpeg	Jun 4, 2024
<input type="checkbox"/>	Block Statue of Nes-Amun.jpg	82.76 KB	image/jpeg	Jun 4, 2024
<input type="checkbox"/>	Eye of Horus Amulet.jpg	28 KB	image/jpeg	Jun 4, 2024
<input type="checkbox"/>	Statue of Isis Nursing Horus.jpg	133.93 KB	image/jpeg	Jun 4, 2024

Block Statue of Ne...

Name
[Block Statue of Nes-Amun.jpg](#)

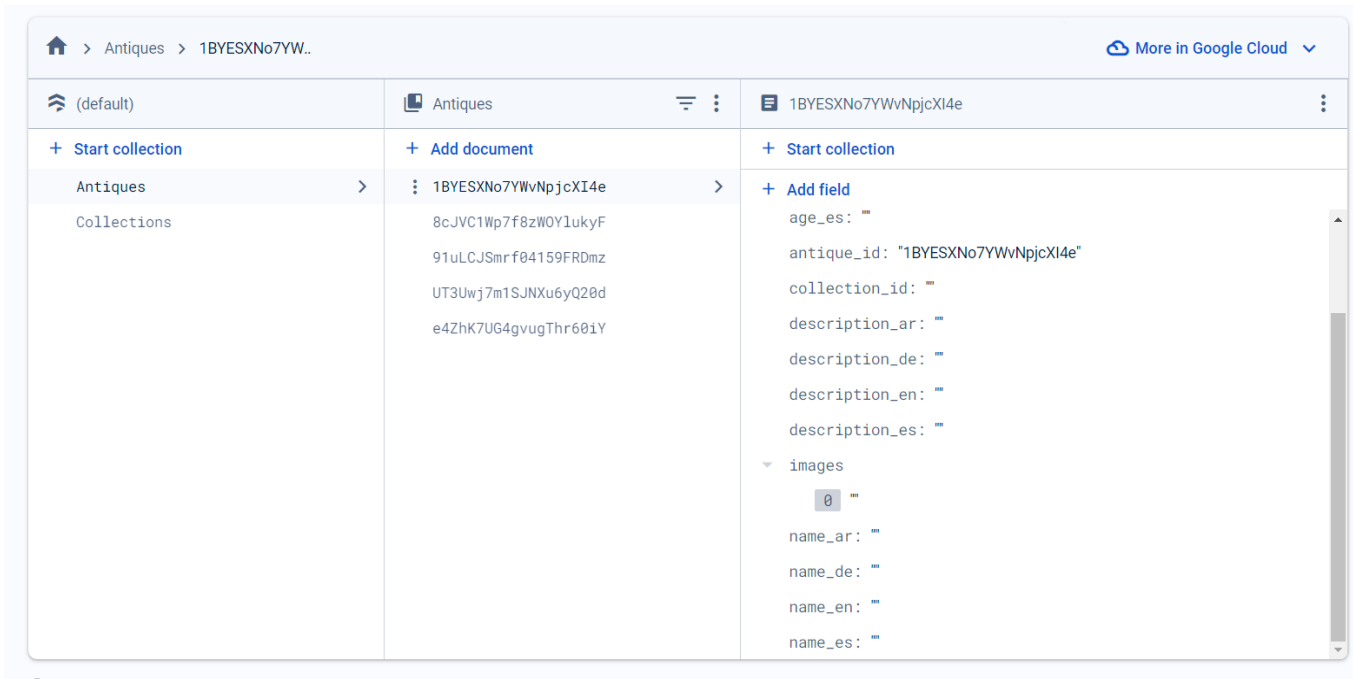
Size
84,750 bytes

Type
image/jpeg

Created
Jun 4, 2024, 4:44:28 PM

Firebase storage contains EVERY image used in the whole application

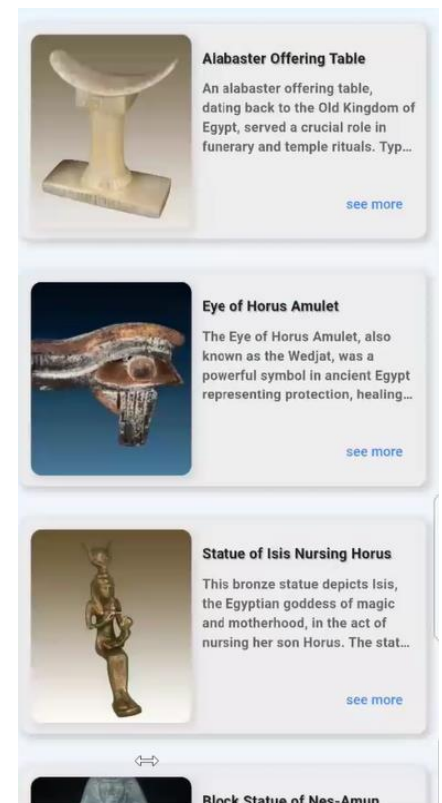
Firestore Database



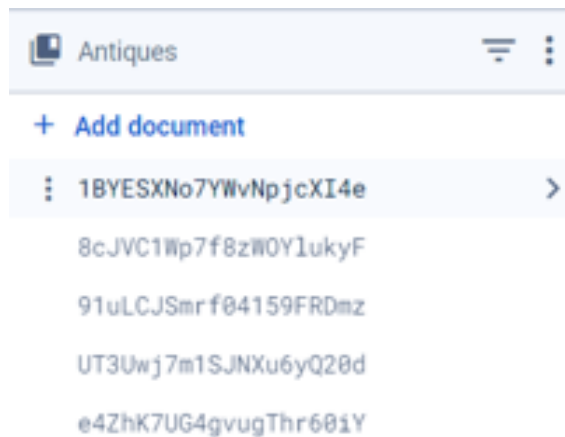
At the very right contains Start collection which functions to start collections



or sections and add in it such as these:

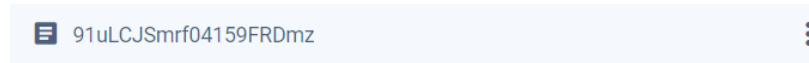


In the middle we have add document which has unique id for every document and is used to be called upon to showcase its components for antiques



At the very last we have adding fields which functions to add components in

The document like this:



age_ar:age info in arabic

+ Start collection

age_de:age info in deutsch

+ Add field

age_en:age info in english

age_ar: "العصر المتأخر، 332-664 قبل الميلاد"

age_de: "Spätzeit, 664-332 v.Chr."

age_en: "Late Period, 664-332 BCE" (string)

age_es: "Período Tardío, 664-332 a.C."

antique_id: "91uLCJSmrf04159FRDmz"

collection_id: "1"

description_ar: "تميمة عين حورس، المعروفة أيضاً بالودجات، كانت رمزاً قوياً في مصر القديمة. يمثل الحماية، الشفاء، والاستعادة. كانت هذه التalism تصنع عادة من الفينيس وغالباً ما تتضمن مزيجاً من عناصر عين الإنسان والصقر، واستخدمت على نطاق واسع في الحياة وفي طقوس الدفن لتوفير الحماية وتعزيز الصحة وإعادة الولادة."

description_de: "Das Amulett des Horusauges, auch bekannt als Wedjat, war ein mächtiges Symbol im alten Ägypten, das Schutz, Heilung und Wiederherstellung darstellte. Häufig aus Favence gefertigt und oft mit einer"

Antique Age:

Late Period, 664-332 BCE

age_es:age info in Spanish

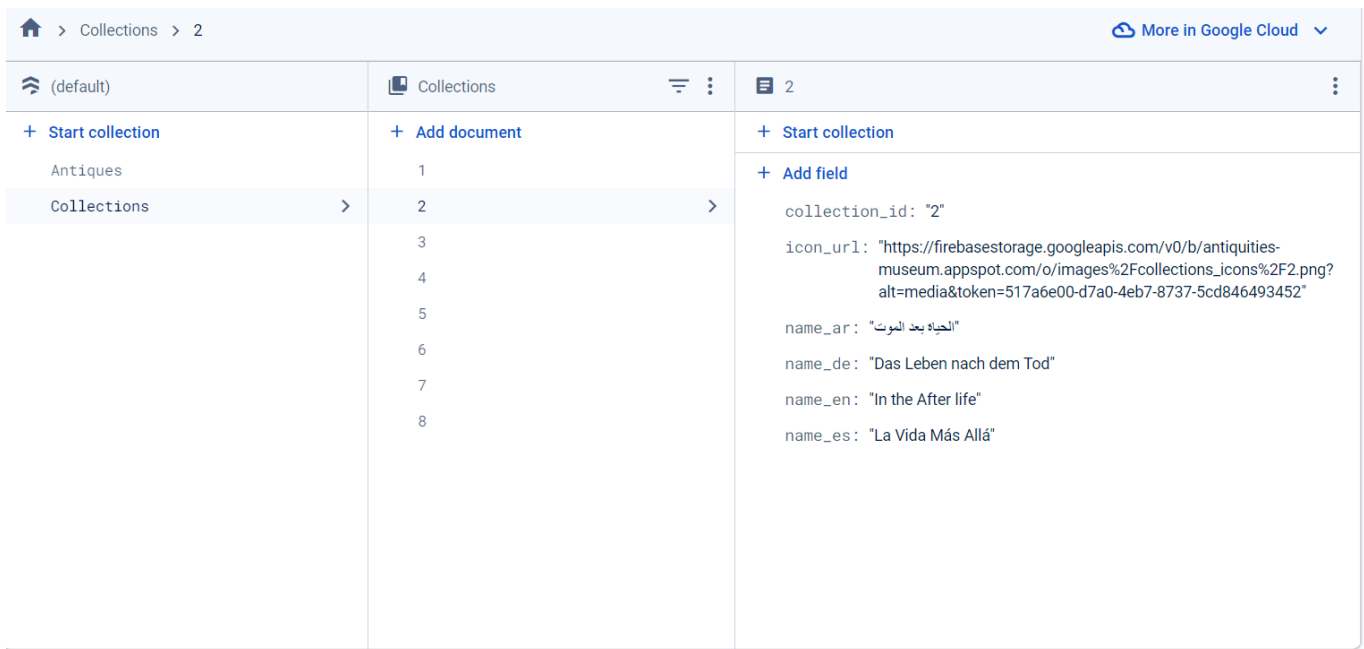
antique_id: unique id for

the antique

collection_id: location

description_ar:Description in Arabic

description_de:Description in deutsch



For Collections which is this:

Add document is used to add these artifact

Groups

Collection_id: location of document

Icon_url:image url

name_ar:name in Arabic

name_de:name in Deutch

name_en:Name in English

name_es:Name in Spanish



5.3: Chatbot Implementation

What is Voiceflow?

Voiceflow is a collaborative design and prototyping tool for creating voice and conversational applications, such as those for Amazon Alexa, Google Assistant, and custom chatbots. It provides a visual interface that allows users to design, prototype, and iterate on voice and chat experiences without needing extensive coding knowledge.

Voiceflow is used by designers, developers, and product managers to create engaging voice and chat experiences efficiently.

How voiceflow works?

Voiceflow works by providing a user-friendly, visual interface that allows users to design, prototype, and deploy voice and conversational applications. Here's a step-by-step overview of how Voiceflow works:

1. **Project Creation:** Users start by creating a new project for their voice or chat application. They can choose from various templates or start from scratch.
2. **Visual Design Interface:** The core of Voiceflow is its drag-and-drop interface, where users can add and connect different elements to build conversational flows. These elements include:
 - **Steps:** Basic building blocks such as speak, listen, choice, and prompt steps.
 - **Integrations:** Pre-built blocks for integrating external APIs, databases, and other services.
 - **Logic:** Elements like conditions, loops, and variables to add logic and interactivity to the conversation.

3. Prototyping: Users can test their designs in real-time within the Voiceflow platform. This allows for immediate feedback and iterative improvement without the need to deploy the application.
4. Collaboration: Voiceflow supports team collaboration, enabling multiple users to work on the same project simultaneously. Team members can leave comments, suggest changes, and track the project's progress.
5. Integrations and API Connections: Users can connect their voice or chat applications to various APIs and services, allowing for more dynamic and interactive experiences. For example, integrating with a weather API to provide real-time weather updates.
6. Testing and Debugging: Voiceflow provides tools for thorough testing and debugging of the conversational flows. Users can simulate different scenarios and ensure the application behaves as expected.
7. Deployment: Once the design and testing are complete, Voiceflow simplifies the deployment process. Users can publish their applications directly to platforms like Amazon Alexa and Google Assistant, or export the code for custom deployment.
8. Iteration and Updates: After deployment, users can continue to update and improve their applications based on user feedback and analytics. Voiceflow makes it easy to iterate and deploy updates quickly.

The benefits of voiceflow over other chatbots

Voiceflow offers a range of benefits that make it a valuable tool for designing and developing conversational applications. These benefits include:

1. User-Friendly Interface:

- Visual Design: The drag-and-drop interface simplifies the design process, allowing users to create conversational flows without needing extensive coding knowledge.
- Accessibility: Suitable for both technical and non-technical users, enabling a wider range of individuals to participate in the development process.

2. Rapid Prototyping and Iteration:

- Real-Time Testing: Allows for immediate testing and feedback, facilitating quick iterations and improvements.
- Efficient Development: Streamlines the creation process, reducing the time and effort required to develop and refine conversational applications.

3. Collaborative Environment:

- Team Collaboration: Supports multiple users working on the same project simultaneously, with features for commenting, change suggestions, and progress tracking.
- Centralized Platform: Provides a single platform for all team members, improving communication and coordination.

4. Integration Capabilities:

- API and Service Integrations: Easily connect applications to various APIs, databases, and external services, enhancing functionality and interactivity.
- Extensibility: Allows for the incorporation of dynamic content and real-time data, creating more engaging user experiences.

5. Simplified Deployment:

- Direct Publishing: Facilitates the deployment of applications to platforms like Amazon Alexa and Google Assistant directly from Voiceflow.

- Code Export: Provides options to export code for custom deployment scenarios, offering flexibility in application distribution.

6. Comprehensive Testing and Debugging:

- Scenario Simulation: Enables thorough testing of different conversational scenarios, ensuring the application performs as expected.

- Debugging Tools: Offers tools to identify and resolve issues during the development process, improving overall application quality.

7. Continuous Improvement:

- User Feedback Integration: Allows for ongoing updates and enhancements based on user feedback and analytics.

- Adaptability: Supports continuous iteration, ensuring the application remains up-to-date and responsive to user needs.

8. Educational Resources and Support:

- Learning Materials: Provides tutorials, documentation, and community support to help users get the most out of the platform.

- Customer Support: Offers assistance guidance to users.

Voiceflow implementation in flutter

1-Add Necessary Dependencies

Add HTTP Package: Add the http package to your pubspec.yaml file for making API requests.

yaml

```
dependencies:  
  flutter:  
    sdk: flutter  
  http: ^0.13.4
```

2-Create a Service for Voiceflow API Interaction

VoiceflowService: Create a Dart class to handle interactions with the Voiceflow API.

```
import 'dart:convert';
```

```
import 'package:http/http.dart' as http;
```

```
class VoiceflowService {
```

```
  final String apiKey;
```

```
  final String versionId;
```

```
  VoiceflowService({required this.apiKey, required this.versionId});
```

```
  Future<Map<String, dynamic>> sendMessage(String message, String  
userId) async {
```

```
    final url = 'https://general-  
runtime.voiceflow.com/state/${versionId}/user/${userId}/interact';
```

```
    final headers = {
```

```
      'Authorization': apiKey,
```

```
      'Content-Type': 'application/json',
```

```

    };

    final body = jsonEncode({
      'request': {
        'type': 'text',
        'payload': message,
      },
    });

    final response = await http.post(Uri.parse(url), headers: headers,
    body: body);

    if (response.statusCode == 200) {
      return jsonDecode(response.body);
    } else {
      throw Exception('Failed to send message to Voiceflow');
    }
  }
}

```

5.3: Graphical User Interface Implementation

GUI:

A GUI (graphical User Interface) is a system of interactive visual components for computer software. A GUI displays objects that convey information, and represent

actions that can be taken by the user. The objects change color, size or visibility when

the user interacts with them.

GUI overview:

A GUI includes GUI objects, like icons, cursors, and buttons. These graphical elements are sometimes enhanced with sounds, or visual effects

like transparency and drop shadows. Using these objects, a user can use the computer without having to know commands.

What are the elements of a GUI?

To make a GUI as user-friendly as possible, there are different elements and objects

that the user use to interact with the software. Below is a list of each of these with a brief description.

- Button - A graphical representation of a button that performs an action in a program when pressed
- Dialog box - A type of window that displays additional information, and asks a user for input.
- Icon - Small graphical representation of a program, feature, or file.
- Menu - List of commands or choices offered to the user through the menu bar.
- Menu bar - Thin, horizontal bar containing the labels of menus.
- Ribbon - Replacement for the file menu and toolbar that groups programs activities together.

- Tab - Clickable area at the top of a window that shows another page or area.
- Toolbar - Row of buttons, often near the top of an application window that controls software functions.
- Window - Rectangular section of the computer's display that shows the program currently being used.

What are the benefits of GUI?

A GUI is more user-friendly than a text-based command-line interface, such as MS-DOS, or the shell of Unix-like operating systems.



Figure 7: Splash screen

Part 1: Splash screen

This page is the main interface of the system, which will appear in the beginning of the interaction between the user and the system



Figure 8: Language Screen

Part 3: Language Screen

This is the Language configuration screen page that is displayed after the splash screen to choose desired language from

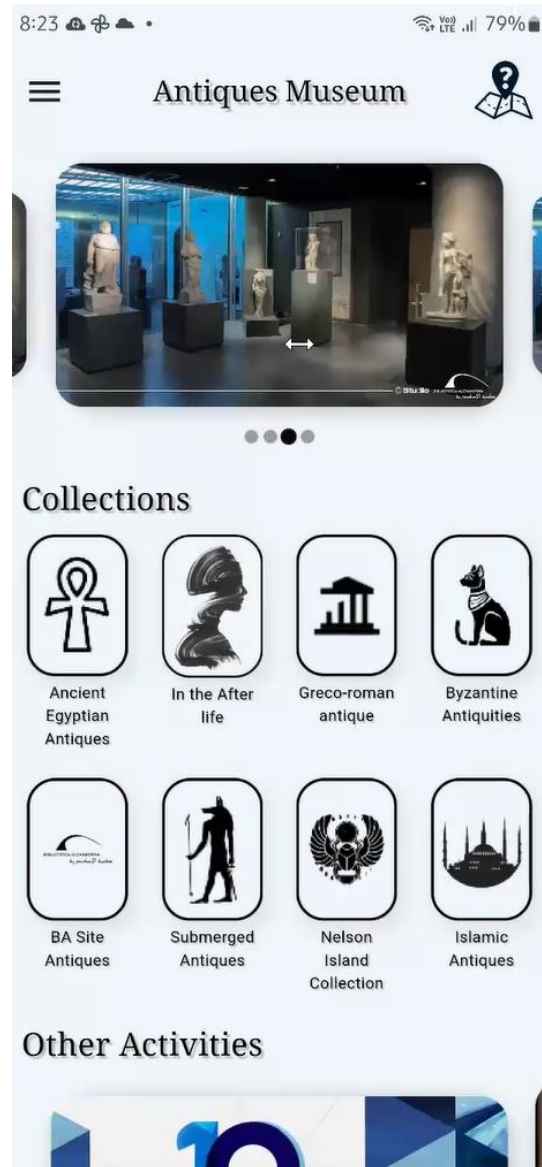


Figure 9: Home Screen

Part 3: Home Screen

The Home Screen is where the user can be directed to any part of the app from



Figure 10: Deutch Home Screen

Preview of Home screen in Deutch



Figure 11: Spanish Home screen

Preview of Home screen in Spanish



Figure 12: Arabic Home screen

Preview of Home screen in Arabic

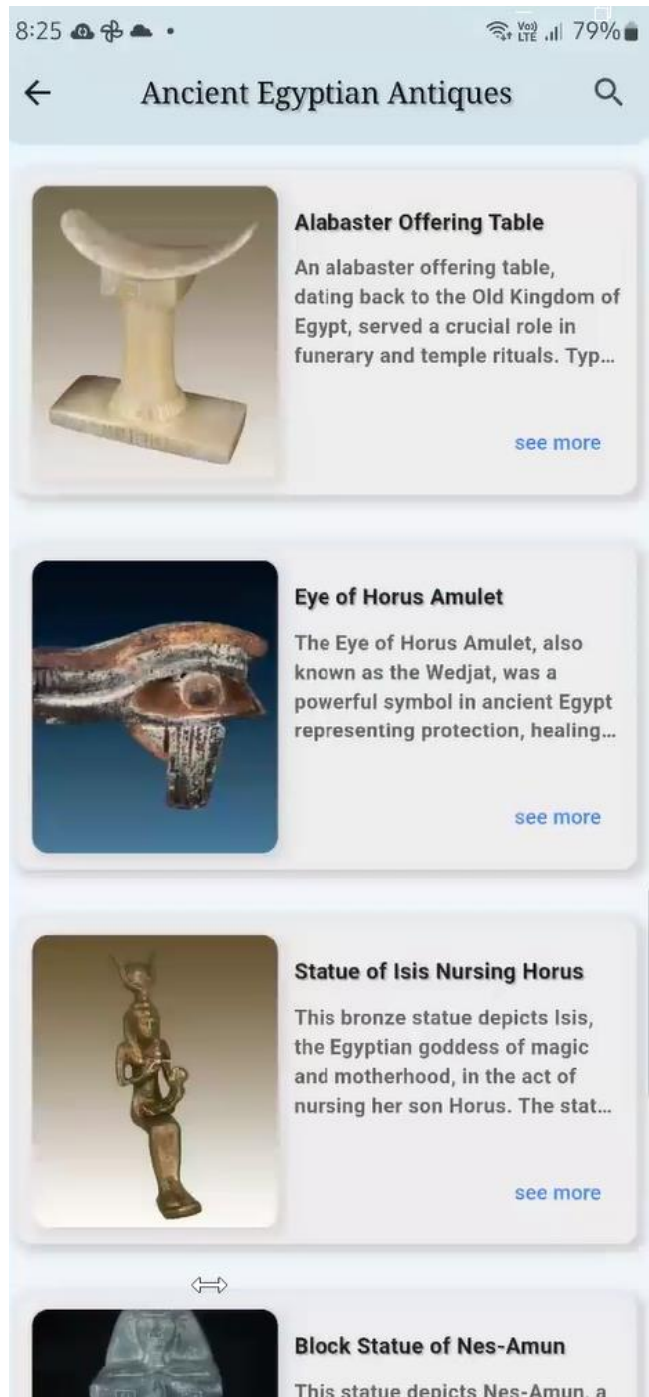


Figure 13: Collection Screen

Part 4: Collection Screen

Showcases the collection of antiquities in a room clickable icons of each antique

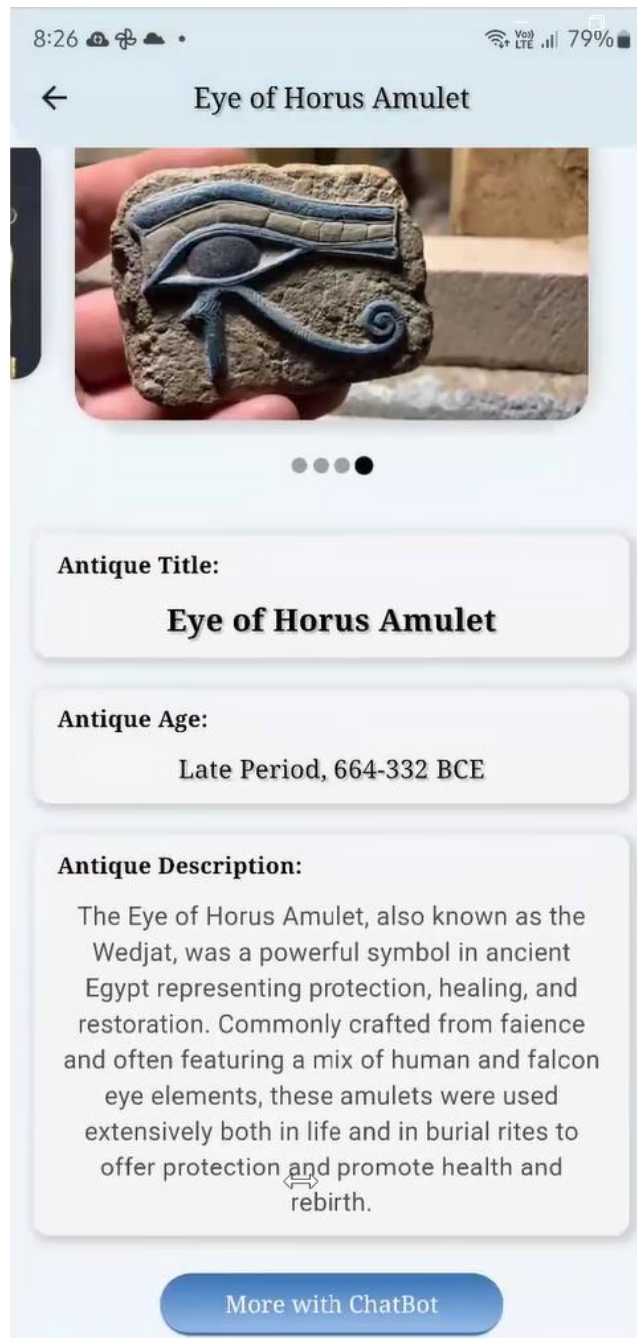


Figure 14: Antique Description Screen

Part 5: Antique Description Screen

This page give information about the antiques along with a button for chatbot

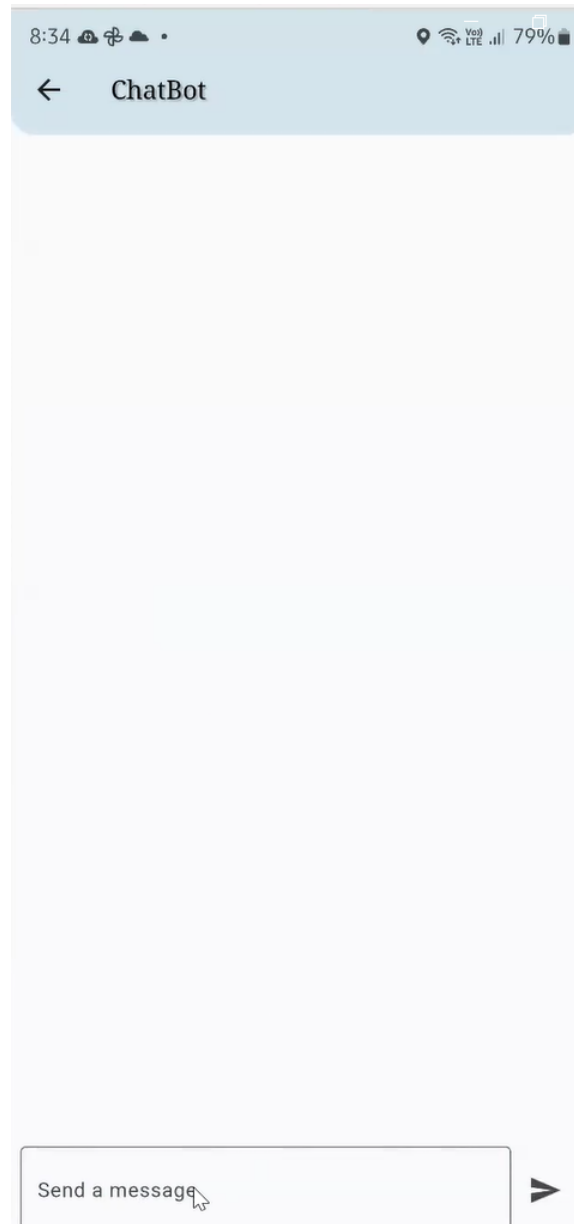


Figure 15: Chatbot Screen

Part 6: Chat bot Screen

This page provides user with a chatbot for him/her to ask more information to satisfy his/her hunger for knowledge

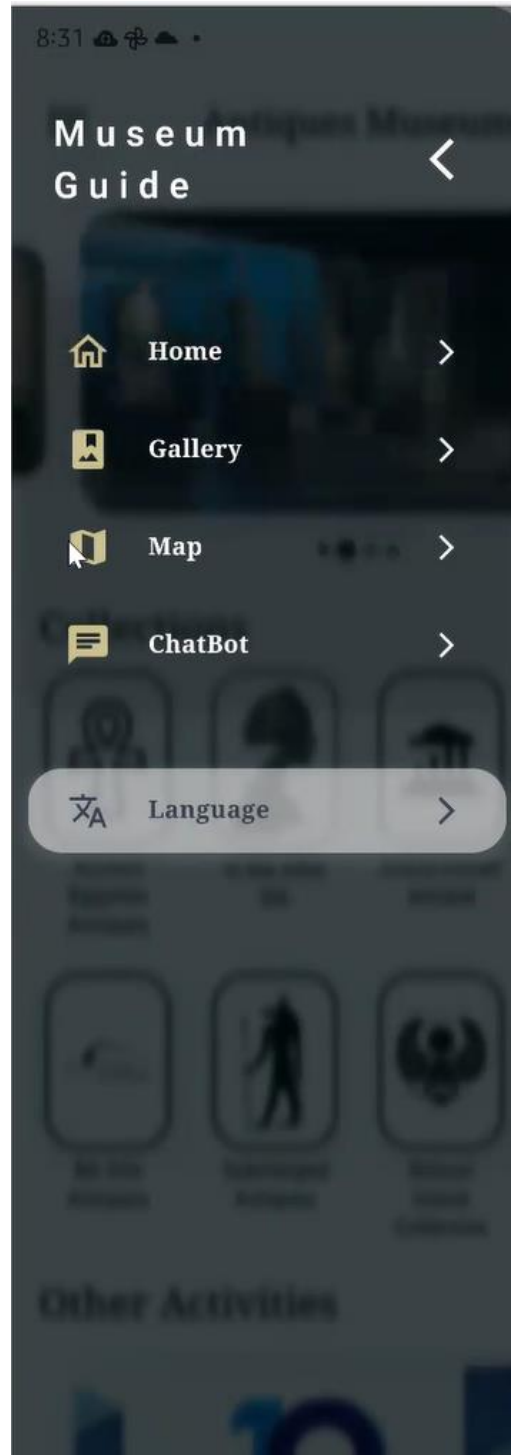


Figure 16: Menu

Part 7: Menu

Showcases a list of directories in the app for the user to navigate through



Figure 17: Gallery

Part 8: Gallery Screen

This page provides images for every antique in the msueum

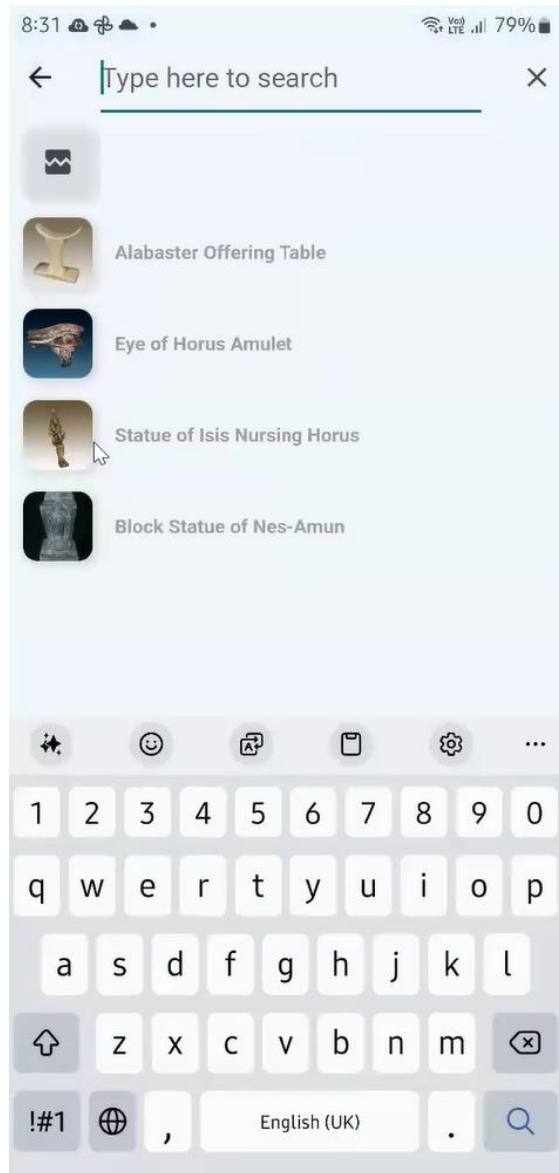


Figure 18: Search

Part 9: Search

This page provides user with a search function to search for specific antique in mind

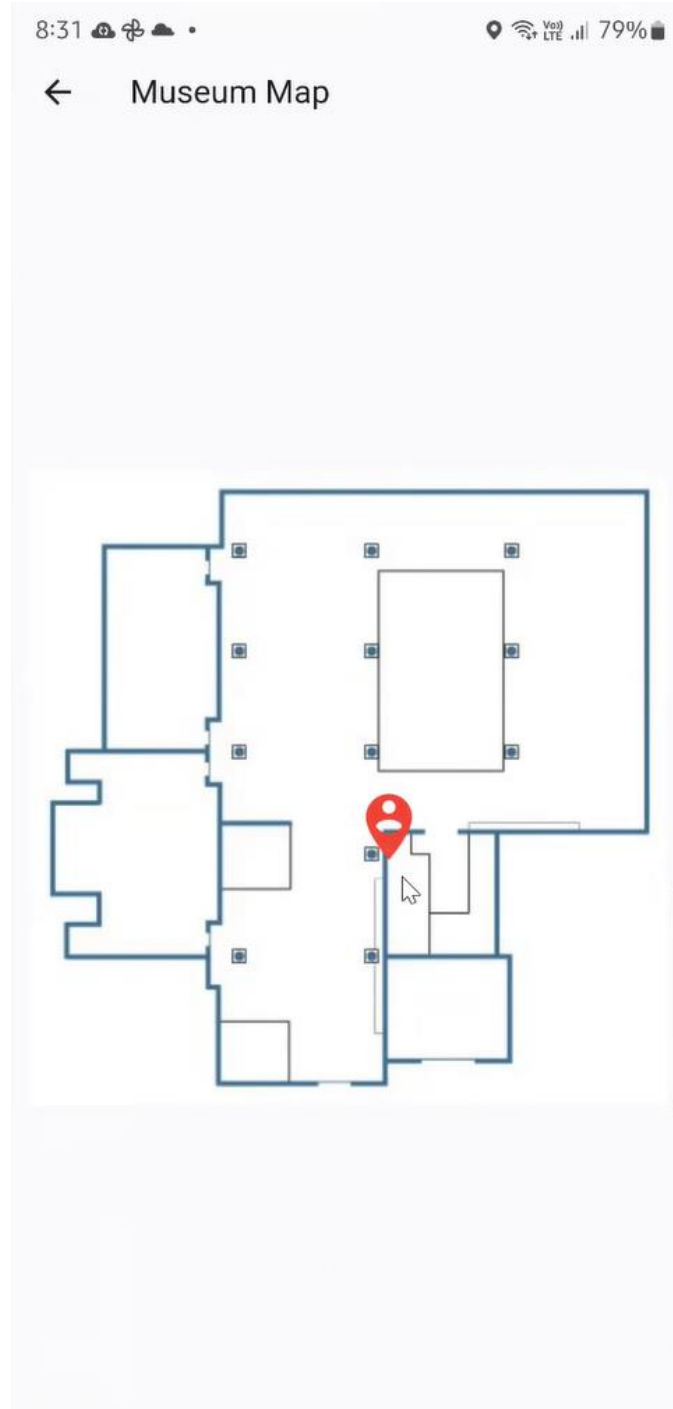


Figure 19: Map screen

Part 10: Map

This is the map page where user is provided with his/her real-time co-ordinates to reach desired destination in the museum

CHAPTER SIX: SYSTEM TESTING

6.1: Introduction

The chapter begins with an in-depth code explanation, detailing the design and logic behind key components of the software. This section aims to give readers a thorough understanding of how the system is constructed and how various modules interact with each other. Following the code explanation, we move into the system testing phase, where various testing methodologies and scenarios are employed to identify and rectify any defects or issues within the system.

6.2: Packages Explanation

Packages

```
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';

import 'app_screens/splash_screen.dart';
import 'firebase_options.dart';
import 'package:flutter_localization/flutter_localization.dart';

import 'package:geolocator/geolocator.dart';
import 'package:permission_handler/permission_handler.dart';

import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart' show FirebaseOptions;
```

1. **import 'package:firebase_core/firebase_core.dart';**
 - Imports the `firebase_core` package, which is necessary to initialize Firebase in a Flutter app.
2. **import 'package:flutter/material.dart';**
 - Imports the Flutter framework's material design library, which provides a rich set of widgets for building UI.
3. **import 'app_screens/splash_screen.dart';**
 - Imports the `SplashScreen` widget from a local file in the `app_screens` directory.
4. **import 'firebase_options.dart';**
 - Imports a local file containing Firebase configuration options specific to the platform.
5. **import 'package:flutter_localization/flutter_localization.dart';**
 - Imports a package for handling localization, allowing the app to support multiple languages.
6. **import 'package:geolocator/geolocator.dart';**
 - provides functionalities to access the location of the device. It allows you to retrieve the current position of the device, listen to location updates, and check or request location permissions.
7. **Import 'package:permission_handler/permission_handler.dart';**
 - A Flutter plugin to handle permissions. It helps in checking the status of various permissions (e.g., location, camera, microphone) and requesting them if
8. **import 'package:flutter/material.dart';**
 - Provides the fundamental building blocks for creating a Flutter app with the Material Design visual language. It includes various widgets, themes, and tools for designing user interfaces.
9. **import 'package:firebase_core/firebase_core.dart' show FirebaseOptions;**
 - Initializes and configures Firebase for a Flutter app. The `FirebaseOptions` class is used to specify Firebase configuration parameters.

6.3: Code Explanation

1.Main.dart

main Function

```
Future<void> main() async {  
  WidgetsFlutterBinding.ensureInitialized();  
  await Firebase.initializeApp(  
    options: DefaultFirebaseOptions.currentPlatform,  
  );  
  runApp(const MyApp());  
}
```

6. Future<void> main() async { ... }:

- The main function is the entry point of the Flutter application. It is marked async to allow for asynchronous operations.

7. WidgetsFlutterBinding.ensureInitialized();:

- Ensures that the Flutter framework is fully initialized before any other operations are performed.

8. await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);:

- Initializes Firebase with platform-specific options defined in firebase_options.dart.

9. runApp(const MyApp());:

- Launches the Flutter application by running the MyApp widget.

MyApp Class

```
class MyApp extends StatefulWidget {  
  final local;  
  const MyApp({super.key, this.local});  
  
  @override  
  State<MyApp> createState() => _MyAppState();  
}
```

10. **class MyApp extends StatefulWidget { ... }:**

- Defines the main application widget as a stateful widget, allowing it to maintain state over its lifetime.

11. **final local;:**

- A final variable (without a specified type) presumably for localization settings. It is part of the MyApp class.

12. **const MyApp({super.key, this.local});:**

- Constructor for the MyApp class, which can optionally take a localization setting.

13. **State<MyApp> createState() => _MyAppState();:**

- Creates the mutable state for MyApp by returning an instance of _MyAppState.

_MyAppState Class

14. **class _MyAppState extends State<MyApp> { ... }:**

- Defines the state class for MyApp, which holds the mutable state for the widget.

15. **@override Widget build(BuildContext context) { ... }:**

- The build method is overridden to define the UI of the app.

build Method

- 16. **return MaterialApp(...);**
 - Returns a MaterialApp widget, which is the root of the application.
- 17. **debugShowCheckedModeBanner: false;**
 - Disables the debug banner in the top right corner of the app.
- 18. **title: 'Antiquities Museum';**
 - Sets the title of the app.
- 19. **theme: ThemeData(...);**
 - Defines the theme of the app using ThemeData.

Theme Customization

- 20. **colorScheme: ColorScheme.fromSeed(seedColor: const Color(0xFF1F8FD)),**
 - Creates a color scheme based on a seed color, providing a consistent color palette.
- 21. **useMaterial3: true;**
 - Enables Material Design 3, the latest version of Google's material design guidelines.
- 22. **elevatedButtonTheme: ElevatedButtonThemeData(...);**
 - Customizes the theme for ElevatedButton widgets.

Elevated Button Theme Customization

- 23. **style: ButtonStyle(...);**
 - Defines the style for elevated buttons.
- 24. **shape: MaterialStateProperty.all<RoundedRectangleBorder>(...);**
 - Sets the shape of the buttons to have rounded corners.
- 25. **padding: MaterialStateProperty.all(...);**
 - Sets the padding for the buttons.
- 26. **textStyle: MaterialStateProperty.all(...);**
 - Sets the text style for the button labels.
- 27. **backgroundColor: MaterialStateProperty.all<Color>(...);**

- Sets the background color for the buttons.

28. **foregroundColor: MaterialStateProperty.all<Color>(...),:**

- Sets the text color for the buttons.

29. **home: const SplashScreen(),:**

- Sets the SplashScreen widget as the home screen of the app.

2.Languages Page

```
class LanguagesScreen extends StatefulWidget {
  final isStart;

  const LanguagesScreen({super.key, required this.isStart});

  @override
  State<LanguagesScreen> createState() => _LanguagesScreenState();
}

body: Container(
  width: MediaQuery.of(context).size.width,
  decoration:isStart ? const BoxDecoration(
    image: DecorationImage(
      image: AssetImage("assets/images/comp/splash_bg.png"), // Path to
your image asset
      fit: BoxFit.fitHeight, // This will fill the screen
    ),
  ):null,
  child: Column(
    children: [
      if(isStart)
        const SizedBox(height: 80),
        Text(isStart ?"Select Language" : (appData.lang == 'ar'? 'اختر اللغة' :
appData.lang == 'es'? 'Seleccionar idioma' : appData.lang == 'de'? 'Sprache
auswählen' : 'Select Language'),
        style: const TextStyle(
          fontSize: 30,
          fontFamily: 'Serif',
          color: Colors.black,
          shadows: [
            Shadow(
              offset: Offset(2.0, 2.0), // Offset for shadow position
              blurRadius: 3.0, // Blur radius for shadow
              color: Colors.white70, // Shadow color
            )
          ]
        ),
      ],
    ),
  ),
)
```

```

        ),
      ],
    ),
  ),
  const SizedBox(height: 90),
  Center(
    child: Wrap(
      spacing: 40,
      runSpacing: 70,
      children: [
        LanguageCard(
          name: 'English',
          imagePath: 'assets/images/comp/flags/english.png',
        ), // English
        LanguageCard(
          name: 'العربية',
          imagePath: 'assets/images/comp/flags/arabic.png',
        ), // Arabic

        LanguageCard(
          name: 'español',
          imagePath: 'assets/images/comp/flags/spanish.png',
        ), // Spanish
        LanguageCard(
          name: 'Deutsch',
          imagePath: 'assets/images/comp/flags/germany.png',
        ), // Germany
      ],
    ),
  ),
],
),),
);
}
}

class LanguageCard extends StatefulWidget {
  final String name;
  final String imagePath;

  LanguageCard({required this.name, required this.imagePath});

  @override
  _LanguageCardState createState() => _LanguageCardState();
}

```

```

}

class _LanguageCardState extends State<LanguageCard> {
  @override
  Widget build(BuildContext context) {
    return InkWell(
      onTap: () async {
        print("${widget.name} tapped");
        if(widget.name == "English"){
          await saveLanguagePreference("en");
        }
        else if(widget.name == "العربية"){
          await saveLanguagePreference("ar");
        }
        else if(widget.name == "español"){
          await saveLanguagePreference("es");
        }
        else if(widget.name == "Deutsch"){
          await saveLanguagePreference("de");
        }
        Navigator.of(context).pushReplacement(MaterialPageRoute(builder: (_) =>
MyApp(local: appData.lang)));
      },
      borderRadius: const BorderRadius.all(Radius.circular(15)),
      child: Container(
        width: 150, // Adjust size as needed
        height: 150,
        decoration: BoxDecoration(
          gradient: const LinearGradient(
            begin: Alignment.topCenter,
            end: Alignment.bottomCenter,
            colors: [Color(0xFFD9E6FF), Color(0xFF1E3050), ],
            stops: [0.0, 1.0],
          ),
        ),
        border: Border.all(
          color: const Color(0xFFB8B8B8),
          width: 1,
        ),
        boxShadow: [
          BoxShadow(
            color: Colors.black.withOpacity(0.25),
            offset: const Offset(0, 4),
            blurRadius: 4,
          ),
        ],
      ),
    );
  }
}

```

```

    ],
    borderRadius: BorderRadius.circular(45),
  ),
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      ClipRRect(
        borderRadius: const BorderRadius.all(Radius.circular(15)),
        child: Image.asset(widget.imagePath, width: 70, height: 70,)
      ),
      const SizedBox(height: 10),
      Text(
        widget.name,
        style: const TextStyle(
          color: Colors.white,
          fontSize: 16,
        ),
      ),
    ],
  ),
);
}
}

```

LanguagesScreen Widget

6. **class LanguagesScreen extends StatefulWidget { ... }:**
 - Defines a stateful widget for the language selection screen.
7. **final isStart;**
 - A final variable that indicates whether this screen is shown at the start of the app. It determines the screen's layout and behavior.
8. **const LanguagesScreen({super.key, required this.isStart});**
 - Constructor for the LanguagesScreen widget, requiring the isStart parameter.
9. **State<LanguagesScreen> createState() => _LanguagesScreenState();**
 - Creates the state for LanguagesScreen.

_LanguagesScreenState Class

10. **class _LanguagesScreenState extends State<LanguagesScreen> { ... }:**
 - Defines the state class for LanguagesScreen.
11. **bool isStart = false;**
 - A boolean variable to hold the initial state of isStart.
12. **@override void initState() { ... }:**
 - Initializes the state. It sets isStart to the value passed from the parent widget.
13. **@override Widget build(BuildContext context) { ... }:**
 - Builds the UI for the screen.

build Method in _LanguagesScreenState

14. **return Scaffold(...);:**
 - Returns a Scaffold widget, which provides the basic material design visual structure.
15. **backgroundColor: isStart ? null : const Color(0xFF1F8FD),:**
 - Sets the background color based on isStart.
16. **appBar: isStart ? null : AppBar(...);:**
 - Conditionally shows an AppBar based on isStart.
17. **body: Container(...);:**
 - Contains the main body of the screen.
18. **decoration: isStart ? const BoxDecoration(...) : null,:**
 - Sets a background image if isStart is true.
19. **child: Column(...);:**
 - Uses a Column to arrange child widgets vertically.

Child Widgets in Column

20. **if(isStart) const SizedBox(height: 80),:**
 - Adds vertical space at the top if isStart is true.
21. **Text(...);:**
 - Displays a text widget for the title "Select Language" with localization support.

22. **const SizedBox(height: 90),:**

- Adds vertical space between the title and the language cards.

23. **Center(...),:**

- Centers the language cards using a Wrap widget.

Wrap Widget for Language Cards

24. **spacing: 40, runSpacing: 70,:**

- Sets the spacing between the language cards.

25. **LanguageCard(...),:**

- Instantiates LanguageCard widgets for each language option.

LanguageCard Widget

26. **class LanguageCard extends StatefulWidget { ... }:**

- Defines a stateful widget for individual language selection cards.

27. **final String name; final String imagePath;:**

- Final variables for the language name and the image path.

28. **LanguageCard({required this.name, required this.imagePath});:**

- Constructor for LanguageCard.

29. **State<LanguageCard> createState() => _LanguageCardState();:**

- Creates the state for LanguageCard.

_LanguageCardState Class

30. **class _LanguageCardState extends State<LanguageCard> { ... }:**

- Defines the state class for LanguageCard.

31. **@override Widget build(BuildContext context) { ... }:**

- Builds the UI for the language card.

build Method in _LanguageCardState

32. **return InkWell(...);:**

- Returns an InkWell widget to make the card tappable.

33. **onTap: () async { ... },:**

- Handles tap events on the language card.

34. **if(widget.name == "English") { await saveLanguagePreference("en"); }:**

- Saves the selected language preference based on the card tapped.

35. **Navigator.of(context).pushReplacement(MaterialPageRoute(builder: (_) => MyApp(local: appData.lang)));:**

- Navigates to the main app screen with the selected language.

Container Inside InkWell

36. **decoration: BoxDecoration(...),:**

- Styles the container with a gradient, border, shadow, and rounded corners.

37. **child: Column(...),:**

- Arranges the language image and name vertically.

38. **ClipRRect(...),:**

- Clips the image to have rounded corners.

39. **Text(...),:**

- Displays the language name below the image.

3.Home Screen

```
class HomeScreen extends StatefulWidget {  
  const HomeScreen({super.key});  
  
  @override  
  State<HomeScreen> createState() => _HomeScreenState();  
}  
  
class _HomeScreenState extends State<HomeScreen> {  
  
  var appName;  
  var title1;  
  var title2;
```

1. State Variables:

- appName, title1, title2: Variables to store app name and titles in different languages.
- _currentIndex: Keeps track of the current index of the carousel.
- banerList: List of URLs for the banner images.
- activitiesList: List of URLs for the activities images.
- collections: List to store collections data.

2. **initState:** Initializes state variables. Sets appName, title1, and title2 based on the app's language setting.

3. **build Method:**

- **MediaQuery:** Retrieves the width of the screen for responsive design.
- **Scaffold:** Provides the basic visual structure (background color, app bar, drawer, body).
- **AppBar:** Customizes the app bar with title and menu icon.
- **Drawer:** User menu.
- **CarouselSlider:** Displays the banner images with auto-play functionality and indicators.
- **SingleChildScrollView:** Allows horizontal scrolling for collections and activities.

Widgets:

1. **CarouselSlider:** Displays a series of images that can be swiped horizontally.
2. **Row:** Displays a row of dots indicating the current slide in the carousel.
3. **GestureDetector:** Wraps around collection items to detect taps and navigate to the CollectionScreen.
4. **Image.network:** Displays images from a URL, with error and loading handling.

Functionality:

- **Localization:** Changes the app name and section titles based on the selected language.
- **Carousel:** Automatically plays and allows users to swipe through images.
- **Refresh Indicator:** Refreshes the screen when pulled down.
- **Dynamic Content:** Displays collections and activities based on provided lists, and allows navigation to detailed screens.

4.Antique screen.dart

```
class AntiqueScreen extends StatefulWidget {
  final antiqueData;

  AntiqueScreen({Key? key, required this.antiqueData}) : super(key: key);

  @override
  _AntiqueScreenState createState() => _AntiqueScreenState();
}

class _AntiqueScreenState extends State<AntiqueScreen> {
  var _antiqueName;
  var _antiqueDes;
  var _antiqueAge;
  var _antiqueImages = [];

  int _currentIndex = 0;

  var _button = '';
  var _title = '';
```

```

var _des = '';
var _age = '';

@override
void initState() {
  super.initState();

  var data = widget.antiqueData;
  _antiqueImages= data['images'];

  if(appData.lang == 'en'){
    _antiqueName = data['name_en'] ?? '';
    _antiqueDes = data['description_en'] ?? '';
    _antiqueAge = data['age_en'] ?? '';

    _button = 'More with ChatBot';
    _title = 'Antique Title: ';
    _des = 'Antique Description: ';
    _age = 'Antique Age: ';
  }
  else if(appData.lang == 'ar'){
    _antiqueName = data['name_ar'] ?? '';
    _antiqueDes = data['description_ar'] ?? '';
    _antiqueAge = data['age_ar'] ?? '';

    _button = 'المزيد مع الدردشة الآلية';
    _title = 'اسم القطعة الأثرية: ';
    _des = 'وصف القطعة الأثرية: ';
    _age = 'تاريخ القطعة الأثرية: ';
  }
  else if(appData.lang == 'es'){
    _antiqueName = data['name_es'] ?? '';
    _antiqueDes = data['description_es'] ?? '';
    _antiqueAge = data['age_es'] ?? '';

    _button = 'Más con ChatBot';
    _title = 'Título de la Reliquia: ';
    _des = 'Descripción de la Reliquia: ';
    _age = 'Época de la Reliquia: ';
  }
  else if(appData.lang == 'de'){
    _antiqueName = data['name_de'] ?? '';

```

```
_antiqueDes = data['description_de'] ?? '';
_antiqueAge = data['age_de'] ?? '';

_button = 'Mehr mit ChatBot';
_title = 'Titel des Antiquitäts:';
_des = 'Beschreibung des Antiquitäts:';
_age = 'Zeitalter des Antiquitäts:';

}
print(_antiqueName);
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: const Color(0xfffff8fd),

    appBar: PreferredSize(
      preferredSize: const Size.fromHeight(60.0), // Set this to whatever height you need
      child: ClipRRect(
        borderRadius: const BorderRadius.vertical(bottom: Radius.circular(20)),
        // Apply rounded corners
        child: AppBar(
          backgroundColor: const Color(0xfffff8fd),
          elevation: 8, // Removes shadow
          title: Text(_antiqueName,
            style: const TextStyle(
              fontSize: 20,
              fontFamily: 'Serif',
              color: Colors.black,
              shadows: [
                Shadow(
                  offset: Offset(1.5, 1.5),
                  blurRadius: 3.0,
                  color: Colors.black26,
                ),
              ],
            ),
          centerTitle: true,
        ),
      ),
    ),
  ),
)
```

Class Definition

1. **AntiqueScreen extends StatefulWidget:**

- This defines a stateful widget called **AntiqueScreen** that can rebuild itself when its state changes.
- **StatefulWidget:** A widget that has mutable state.

Constructor

2. **AntiqueScreen({Key? key, required this.antiqueData}):**

- Constructor with an optional key parameter and a required antiqueData parameter.
- **Key? key:** An optional parameter used to control the widget's unique identity.
- **required this.antiqueData:** A required parameter that provides data about the antique.

3. **@override _AntiqueScreenState createState() => _AntiqueScreenState();**

- Creates the mutable state for this widget.

State Class

4. **class _AntiqueScreenState extends State<AntiqueScreen>:**

- Defines the mutable state class **_AntiqueScreenState**.

State Variables

5. **var _antiqueName, _antiqueDes, _antiqueAge;**

- Variables to store the antique's name, description, and age.

6. **var _antiqueImages = [];**

- A list to store the URLs of the antique's images.

7. **int _currentIndex = 0;**

- Keeps track of the current index of the carousel slider.

8. **var _button, _title, _des, _age;**

- Variables for button text and labels based on the selected language.

initState Method

9. **@override void initState() { super.initState(); ... }**

- Initializes the state of the widget.
- Sets the antique data and localized texts based on the app's language setting.

Localization

10. **if (appData.lang == 'en') { ... } else if (appData.lang == 'ar') { ... }**

- Checks the app's language setting and sets the appropriate texts for each variable.

build Method

11. **@override Widget build(BuildContext context) { ... }**

- Builds the widget tree.

12. **return Scaffold(...);**

- Creates a scaffold, which is a structure for material design visual layout.

AppBar

13. **appBar: PreferredSize(...);**

- Customizes the app bar with preferred size, rounded corners, and styling.

Body

14. **body: SingleChildScrollView(...);**

- Allows vertical scrolling for the content.

Carousel Slider

15. **CarouselSlider(...);**

- Displays a horizontal slider for the antique images.
- **options: CarouselOptions(...);** Configures the slider options such as autoplay, aspect ratio, etc.
- **items: _antiqueImages.map(...).toList();** Maps the images to a list of widgets.

Image Container

16. **Container(...):**

- Wraps each image with padding, decoration, and error/loading handling.

Image Indicators

17. **Row(...):**

- Displays indicators for the current image in the carousel.

Antique Details

18. **Container(...):**

- Displays the antique's title, age, and description in separate containers with styling.

Button

19. **ElevatedButton(...):**

- Displays a button with gradient and rounded corners.

showImageDialog Function

20. **void showImageDialog(BuildContext context, String imageUrl) { ... }**

- Displays a dialog with a full-screen view of the tapped image.

Dialog

21. **showDialog(...):**

- Shows a dialog with blurred background and close button.
- **PhotoView(...):** Displays the image with zoom and pan functionality.

5.Collection Screen.dart

```
class CollectionScreen extends StatefulWidget {
  final String collectionId;
  final String title;

  CollectionScreen({Key? key, required this.collectionId, required this.title}) :
    super(key: key);

  @override
  _CollectionScreenState createState() => _CollectionScreenState();
}

class _CollectionScreenState extends State<CollectionScreen> {
  late Stream<QuerySnapshot> _antiquesStream;
  var lang;

  @override
  void initState() {
    super.initState();
    // Initialize the stream
    lang = appData.lang;

    _antiquesStream = FirebaseFirestore.instance
      .collection('Antiques')
      .where('collection_id', isEqualTo: widget.collectionId)
      .snapshots();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color(0xffF1F8FD),

      appBar: PreferredSize(
        preferredSize: const Size.fromHeight(60.0), // Set this to whatever
height you need
        child: ClipRRect(
          borderRadius: const BorderRadius.vertical(bottom: Radius.circular(20)),
          // Apply rounded corners
          child: AppBar(
            backgroundColor: const Color(0xffF1F8FD),
            elevation: 8,
            centerTitle: true,
            title: Text(widget.title,
```

```

        style: const TextStyle(
          fontSize: 20,
          fontFamily: 'Serif',
          color: Colors.black,
          shadows: [
            Shadow(
              offset: Offset(1.5, 1.5),
              blurRadius: 3.0,
              color: Colors.black26,
            ),
          ],
        ),
      ),
    actions: [
      Padding(
        padding: const EdgeInsets.all(8.0),
        child: IconButton(onPressed: (){
          showSearch(context: context, delegate: SearchScreen());
        }, icon: Icon(Icons.search)),
      )
    ],
  ),
),
),
),

```

- **class CollectionScreen extends StatefulWidget:** This defines a new stateful widget named CollectionScreen. A stateful widget is a widget that maintains state that might change during the lifecycle of the widget.
- **final String collectionId; final String title;** These are the properties of the CollectionScreen class. They are marked as final because their values are set once and never change.
- **CollectionScreen({Key? key, required this.collectionId, required this.title}) : super(key: key);** This is the constructor for CollectionScreen. It initializes the widget with the collectionId and title parameters and passes the optional key parameter to the superclass constructor.

- **@override _CollectionScreenState createState() => _CollectionScreenState();** This method creates the mutable state for this widget. The `_CollectionScreenState` class, defined below, manages the state for `CollectionScreen`.

Class `_CollectionScreenState`

Definition and State Variables

dart

Copy code

```
class _CollectionScreenState extends State<CollectionScreen> {
  late Stream<QuerySnapshot> _antiquesStream;
  var lang;
```

- **class _CollectionScreenState extends State<CollectionScreen>**: This defines the state class `_CollectionScreenState` which extends `State<CollectionScreen>`. This class contains the mutable state for the `CollectionScreen` widget.
- **late Stream<QuerySnapshot> _antiquesStream;**: This declares a late-initialized stream of type `QuerySnapshot`, which will be used to retrieve data from Firestore.
- **var lang;**: This variable will hold the current language setting.

@override void initState(): This method is called once when the state is created. It is used to initialize data that the widget will use.

super.initState();: Calls the `initState` method of the superclass to ensure any inherited initialization is performed.

lang = appData.lang;: Initializes the `lang` variable with the current language setting from `appData`.

antiquesStream = FirebaseFirestore.instance.collection('Antiques').where('collection_id', isEqualTo: widget.collectionId).snapshots(); Initializes `_antiquesStream` to a stream that listens to changes in the Firestore collection `Antiques` where the `collection_id` matches the `collectionId` passed to the widget.

```
onPressed: () {  
  Navigator.push(  
    context,  
    MaterialPageRoute(  
      builder: (context) => AntiqueScreen(antiqueData: data),  
    ),  
  );  
},
```

This part of the code defines an `onPressed` event handler for a `TextButton`. When this button is pressed, it navigates to a new route using the `Navigator.push` method. It pushes a new route onto the navigator's stack, which displays the `AntiqueScreen` widget.

6.Gallery Screen.dart

```
class galleryScreen extends StatefulWidget {  
  
  galleryScreen({Key? key,}) : super(key: key);  
  
  @override  
  _galleryScreenState createState() => _galleryScreenState();  
}
```

galleryScreen

- **Class Declaration:** `galleryScreen` extends `StatefulWidget`.
- **Constructor:**
 - `galleryScreen({Key? key}) : super(key: key);`

- This constructor optionally takes a Key and passes it to the superclass StatefulWidget.
- **createState Method:**
 - `_galleryScreenState createState() => _galleryScreenState();`
 - Creates the mutable state for this widget.

_galleryScreenState

- **Class Declaration:** `_galleryScreenState` extends `State<galleryScreen>`.
- **Member Variables:**
 - `_firestore`: An instance of `FirebaseFirestore` to interact with Firestore.
 - `_antiquesStream`: A `Stream<QuerySnapshot>` to handle the stream of antique data from Firestore.
 - `_collectionsList`: A variable to store the list of collections.
 - `lang`: A variable to store the selected language.

```

class _galleryScreenState extends State<galleryScreen> {
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;

  late Stream<QuerySnapshot> _antiquesStream;
  var _collectionsList;

  var lang;

  @override
  void initState() {
    super.initState();
    // Initialize the stream
    lang = appData.lang;
    _collectionsList = appData.Collections;
    //print("_collectionsList: $_collectionsList");
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color(0xffF1F8FD),
      appBar: PreferredSize(
        preferredSize: const Size.fromHeight(60.0), // Set this to whatever
height you need
        child: ClipRRect(
          borderRadius: const BorderRadius.vertical(bottom: Radius.circular(20)),
          // Apply rounded corners
          child: AppBar(
            backgroundColor: const Color(0xffF1F8FD),
            elevation: 8, // Removes shadow
            title: Text(lang == 'ar'? 'المعرض' : lang == 'es'? 'Galería' : lang
== 'de'? 'Galerie' : "Gallery",
            style: const TextStyle(
              fontSize: 20,
              fontFamily: 'Serif',
              color: Colors.black,
              shadows: [
                Shadow(
                  offset: Offset(1.5, 1.5),
                  blurRadius: 3.0,
                  color: Colors.black26,
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }
}

```

```

    ),),
    actions: [
      Padding(
        padding: const EdgeInsets.all(8.0),
        child: IconButton(onPressed: (){
          showSearch(context: context, delegate: SearchScreen());
        }, icon: const Icon(Icons.search)),
      )
    ],
  ),
),
),
),

```

Lifecycle Methods

initState

- Called when the state is initialized.
- **Initializations:**
 - `lang = appData.lang`; Sets the language from global app data.
 - `_collectionsList = appData.Collections`; Sets the list of collections from global app data.

7.map_Screen.dart

```
class MuseumMap extends StatefulWidget {  
  @override  
  _MuseumMapState createState() => _MuseumMapState();  
}  
  
class _MuseumMapState extends State<MuseumMap> {  
  double _xPosition = 100.0;  
  double _yPosition = 200.0;  
  final double _mapWidth = 386.0; // Set this to the width of your map image  
  final double _mapHeight = 386.0; // Set this to the height of your map image
```

MuseumMap

- **Class Declaration:** MuseumMap extends StatefulWidget.
- **createState Method:**
 - _MuseumMapState createState() => _MuseumMapState();
 - Creates the mutable state for this widget.

_MuseumMapState

- **Class Declaration:** _MuseumMapState extends State<MuseumMap>.

Member Variables

- _xPosition: The x-coordinate of the user's position on the map.
- _yPosition: The y-coordinate of the user's position on the map.
- _mapWidth: The width of the map image.
- _mapHeight: The height of the map image.
- _minLat: The minimum latitude of the GPS bounding box.
- _maxLat: The maximum latitude of the GPS bounding box.
- _minLng: The minimum longitude of the GPS bounding box.
- _maxLng: The maximum longitude of the GPS bounding box.

Lifecycle Methods

initState

- Called when the state is initialized.
- Calls `_checkAndRequestPermissions()` to ensure location permissions are granted.

Permission and Location Handling

_checkAndRequestPermissions

- Checks the location permission status.
- If granted, calls `_getCurrentLocation()` and `_startLocationUpdates()`.
- If denied, requests permission and handles the result

```
Future<void> _checkAndRequestPermissions() async {
  var status = await Permission.location.status;
  if (status.isGranted) {
    _getCurrentLocation();
    _startLocationUpdates();
  } else if (status.isDenied || status.isPermanentlyDenied) {
    var result = await Permission.location.request();
    if (result.isGranted) {
      _getCurrentLocation();
      _startLocationUpdates();
    } else {
      // Handle the case when the user denies the permission
      openAppSettings();
    }
  }
}
```

_getCurrentLocation

- Retrieves the current location using Geolocator.
- Calls `_setBoundingBox` and `_updateUserPosition` with the current latitude and longitude.

```
Future<void> _getCurrentLocation() async {
  Position position = await Geolocator.getCurrentPosition(desiredAccuracy:
LocationAccuracy.high);
  _setBoundingBox(position.latitude, position.longitude);
  _updateUserPosition(position.latitude, position.longitude);
  print("Current latitude: ${position.latitude}");
  print("Current longitude: ${position.longitude}");
}
```

_startLocationUpdates

- Starts listening for location updates.
- Calls `_updateUserPosition` with the new latitude and longitude.

```
void _startLocationUpdates() {
  Geolocator.getPositionStream(
    locationSettings: LocationSettings(
      accuracy: LocationAccuracy.high,
      distanceFilter: 1, // Minimum change in distance (meters) before updates
are sent
    ),
  ).listen((Position position) {
    _updateUserPosition(position.latitude, position.longitude);
    print("New latitude: ${position.latitude}");
    print("New longitude: ${position.longitude}");
    print("===" * 10);
  });
}
```



```

void _setBoundingBox(double lat, double lng) {

    const double distance = 30.0; // 500 meters

    // Calculate latitude bounds (1 degree latitude ~ 111.32 km)
    double latDelta = distance / 111320.0; // meters to degrees
    _minLat = lat - latDelta;
    _maxLat = lat + latDelta;
    print("Museum minLat: $_minLat");
    print("Current maxLat: $_maxLat");

    // Calculate longitude bounds (1 degree longitude ~ varies with latitude)
    double lngDelta = distance / (111320.0 * cos(lat * (pi / 180.0)));
    _minLng = lng - lngDelta;
    _maxLng = lng + lngDelta;
    print("Museum minLng: $_minLng");
    print("Current maxLng: $_maxLng");
}

```

_setBoundingBox

- Calculates the bounding box for the museum's location based on a central latitude and longitude.
- The bounding box is defined by a minimum and maximum latitude and longitude.
- Uses a fixed distance (30 meters in this case) to define the bounds around the current location.

```

• void _updateUserPosition(double lat, double lng) {
•     // Convert GPS coordinates to image coordinates
•     double x = _mapWidth * (lng - _minLng) / (_maxLng - _minLng);
•     double y = _mapHeight * (1 - (lat - _minLat) / (_maxLat - _minLat));
•     // Invert y-axis
•
•     // Ensure the position is within the map boundaries
•     setState(() {
•         _xPosition = x.clamp(0.0, _mapWidth - 48.0); // 48 is the width of
the pin
•         _yPosition = y.clamp(0.0, _mapHeight - 48.0); // 48 is the height of
the pin
•     });
• }

```

_updateUserPosition

- Converts GPS coordinates (latitude and longitude) to image coordinates (x and y) for positioning on the map.
- Uses the bounding box to normalize the coordinates within the map dimensions.
- Ensures the calculated position is within the bounds of the map image.

8.Splash Screen.dart

```
void handleAppData(context) async {
  var lang = await getLanguagePreference();
  //print("lang: $lang");

  if(lang != null){
    appData.lang = lang;
    var isData = await getAppData();
    //print("Collections: ${appData.Collections}");

    if(isData){
      Timer(const Duration(milliseconds: 2500), () {
        Navigator.of(context).pushReplacement(MaterialPageRoute(builder: (_) =>
const HomeScreen()));
      });
    }
  } else {
    Timer(const Duration(milliseconds: 3500), () {
      Navigator.of(context).pushReplacement(MaterialPageRoute(builder: (_) =>
const LanguagesScreen(isStart: true,)));
    });
  }
}
```

handleAppData

- Asynchronously handles initial app data setup.
- Retrieves the user's language preference using `getLanguagePreference()`.
- If a language preference is found:
 - Sets the app's language.
 - Retrieves app data using `getAppData()`.
 - If app data is successfully retrieved, navigates to the HomeScreen after a delay.
- If no language preference is found, navigates to the LanguagesScreen after a delay.

```

@override
Widget build(BuildContext context) {
  var width = MediaQuery.of(context).size.width;
  return Scaffold(
    body: Container(
      width: MediaQuery.of(context).size.width,
      decoration: const BoxDecoration(
        image: DecorationImage(
          image: AssetImage("assets/images/comp/splash_bg.png"), // Path to
your image asset
          fit: BoxFit.fitHeight, // This will fill the screen
        ),
      ),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        Image.asset('assets/images/logos/app_animi_puls_logo.gif', width:
width * .9),
        Image.asset('assets/images/comp/splash_text.png', width: width *
.95),

        const SizedBox(height: 20),
        const CircularProgressIndicator(valueColor:
AlwaysStoppedAnimation<Color>(Color(0xff021930))),
      ],
    ),
  );
}

```

build

- Builds the UI of the splash screen.
- Uses `MediaQuery` to get the screen width.
- Constructs a `Scaffold` with a background image and a centered column of widgets.
- The column contains:
 - An animated logo (`app_animi_puls_logo.gif`).
 - A text image (`splash_text.png`).
 - A `CircularProgressIndicator` to indicate loading.

6.4: System Testing

Software testing is a crucial part of software quality assurance and is a review of the coding, design, and specification. The process of software testing is used to determine the accuracy, completeness and developed quality. Technical investigation known as testing is carried out on behalf of stakeholders with the goal of revealing information about the product's quality in relation to the environment in which it will be used. Software testing, put simply, is the process of determining whether the results obtained match those anticipated and that there are no flaws in the software system. The types of testing we have already used to test our application, such as functional testing, unit testing, will be covered in the following sections, then state test cases we already used to test how our application will behave in the running environment.



Figure 20: splash screen test

After starting the application the splash screen appears, it is the application logo and slogan along with a Museum background that is the main theme for our application.



Figure 21: Language screen test

After the splash screen, the user should be directed to a screen where they can choose their preferred language. This screen will provide multiple language options (e.g., English, Arabic, Spanish, German).

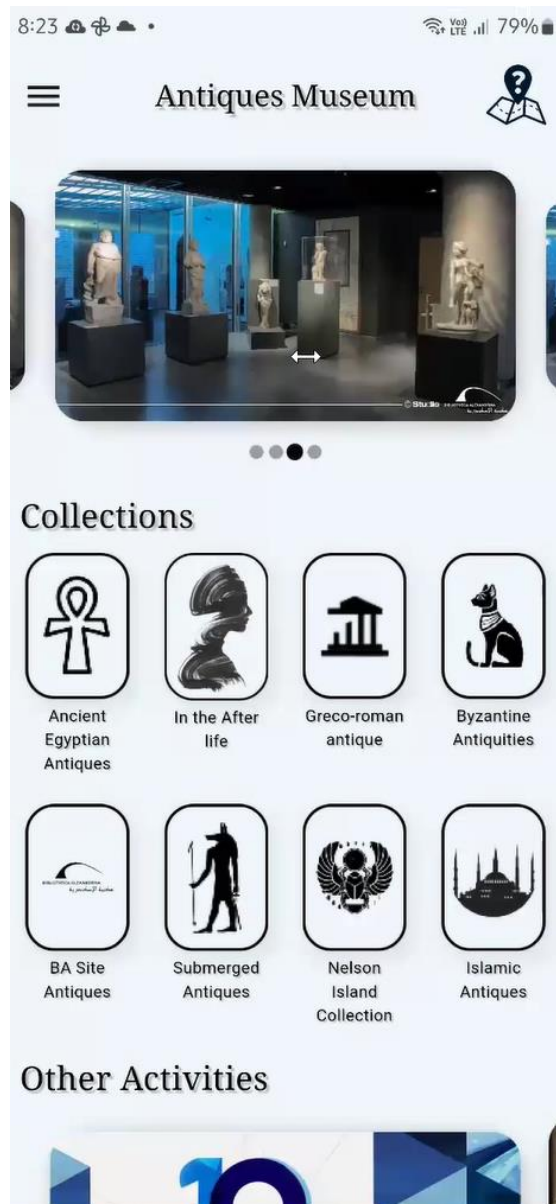


Figure 22: Home Screen test

After selecting the language the user will should directed to Home screen with the language he selected which contains images of the museum, collections of antiques which he can choose from, and other activities

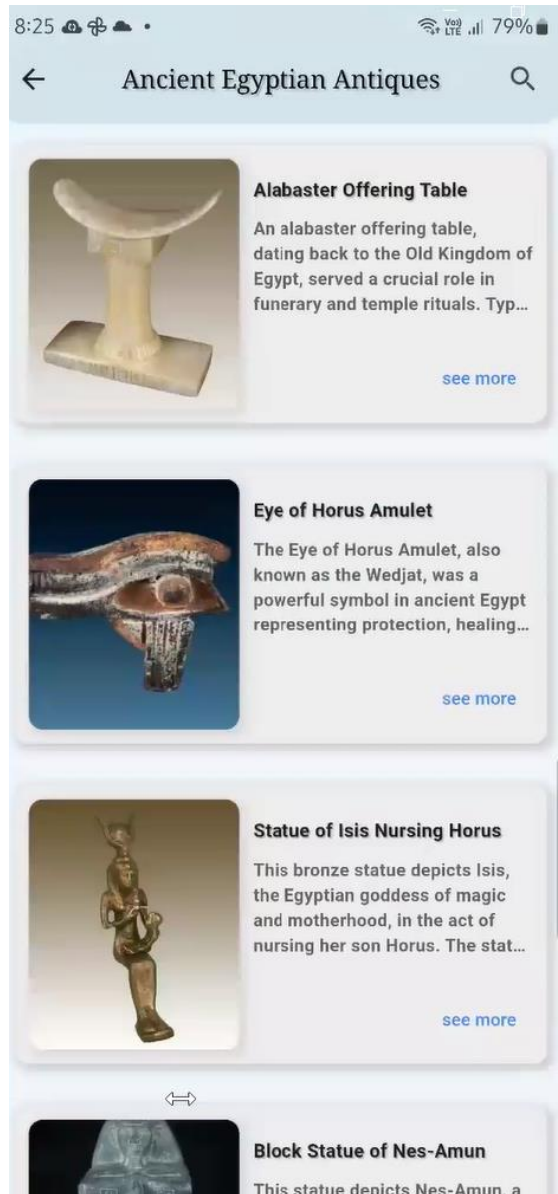


Figure 23: Collection Images test

After he choose one of the collection he will directed to the collection's images and description



Figure 24: Antique description test

When he click on the antique he will directed to description screen of this antique , he also can click on more with chatbot button

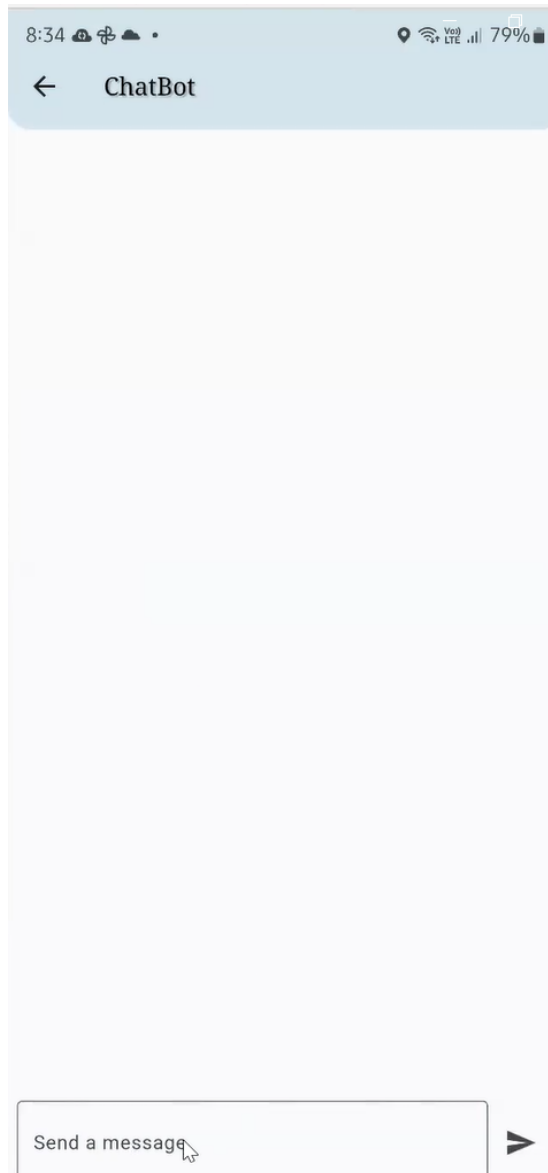
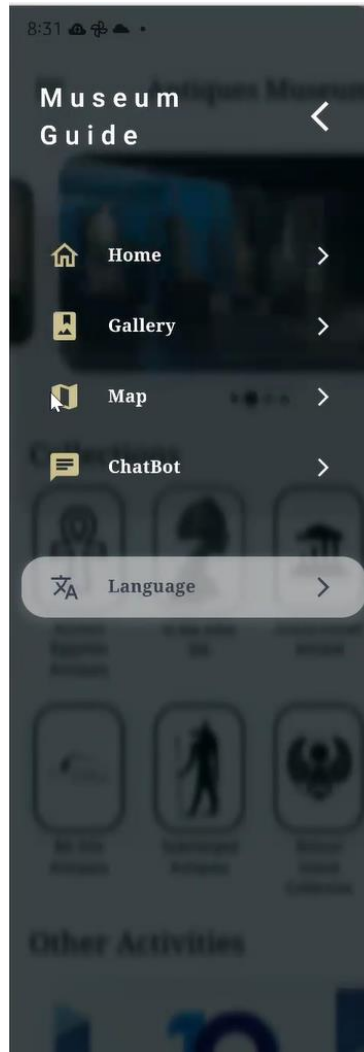
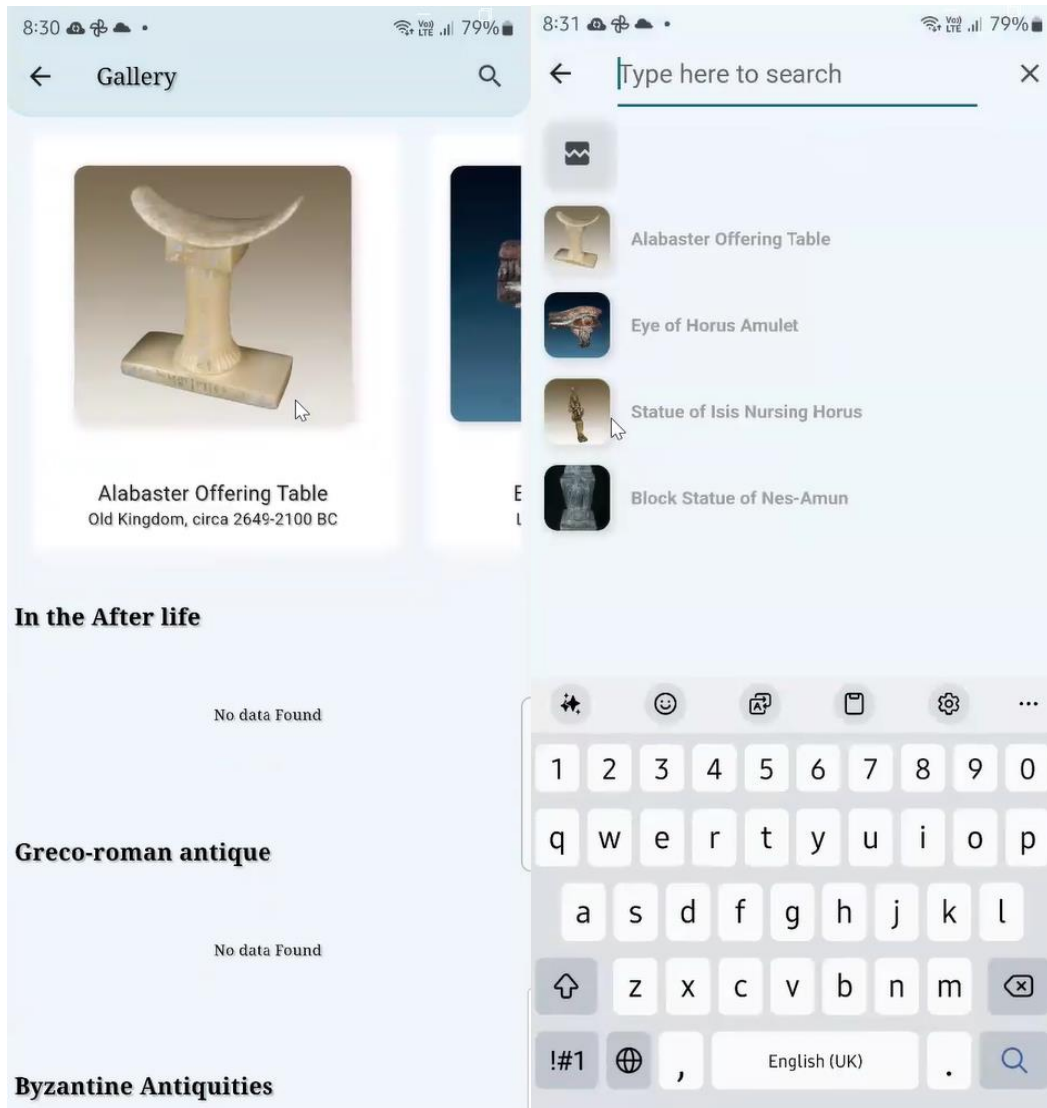


Figure 25: Chatbot Test

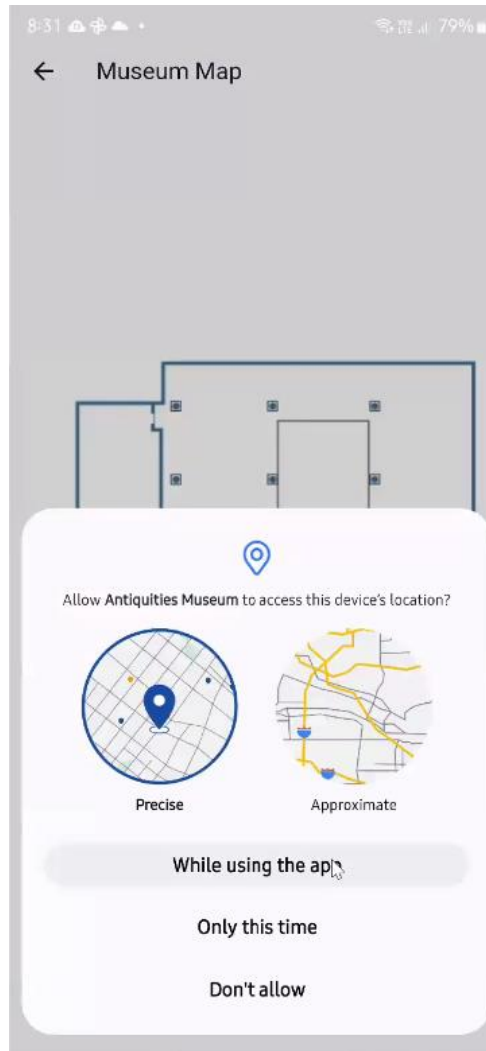
After clicking on the button he will directed to the chatbot screen, he can ask any questions about the museum and antiques.



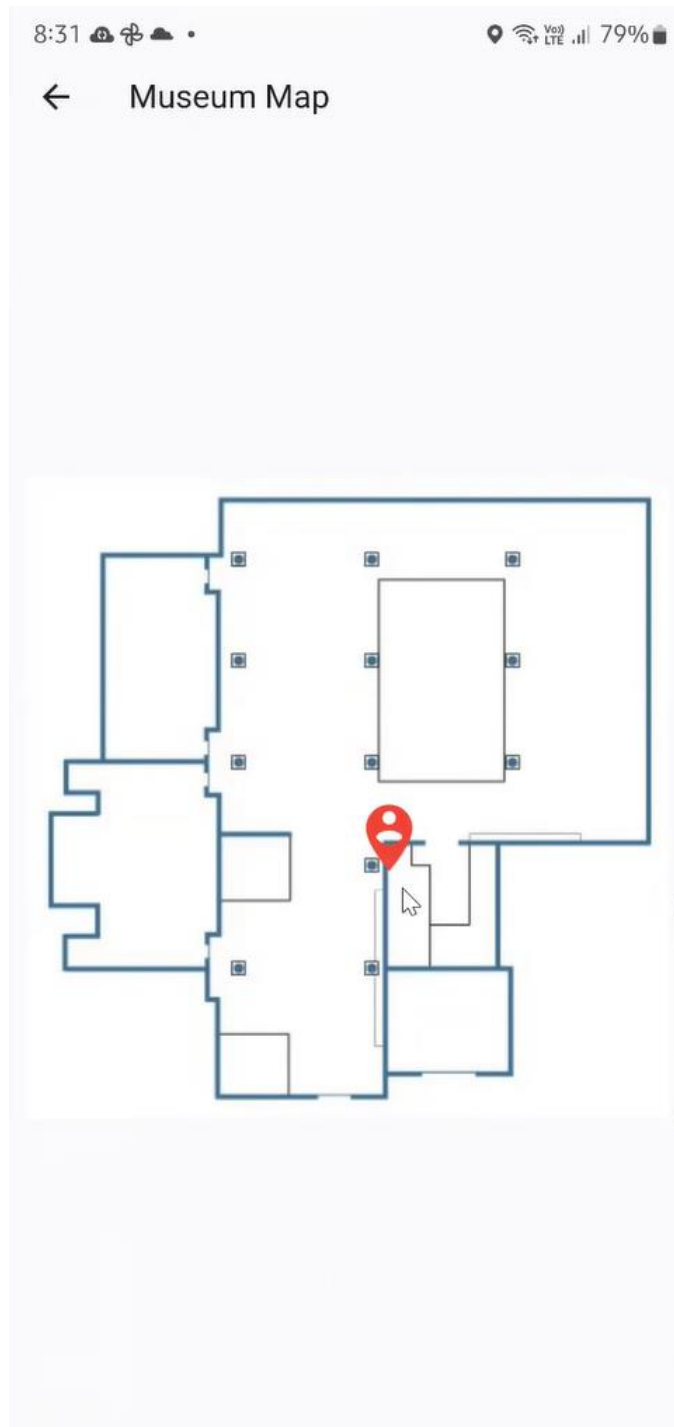
From the menu he can choose from Home , gallery , map , and chatbot



When he chooses gallery it will directed to gallery page that contains all the images of the collection and he can also search about specific image



After he chooses the map he will be directed to the map screen, and the app will ask for the device's location so the user will allow or deny



After the accept of location the app will detect the user's location and start seeing the places in the museum

6.5: Summary

In this chapter, we have delved into the critical aspects of system testing, starting with a detailed explanation of the code. By breaking down the code, we aimed to enhance understanding of the system's architecture and functionality. The subsequent system testing section highlighted the methods and tools used to ensure the software meets all specified requirements and performs reliably under various conditions.

CHAPTER SEVEN: PROJECT CONCLUSION AND FUTURE WORK

7.1: Introduction

In this chapter, we provide a comprehensive conclusion to our graduation project focused on the development of the Antiquities Museum app for the Bibliotheca Alexandrina. Our project aimed to create a digital platform that enhances visitor engagement and education within the museum setting, leveraging modern technology to provide immersive and interactive experiences. Throughout this chapter, we reflect on the strengths and weaknesses of the app, explore potential areas for future enhancement and expansion, and summarize the key findings and contributions of our work.

7.2: Overall Weaknesses

Technical Challenges:

ArcGIS

The system needs to use the ArcGIS program in drawing maps, but there were some difficulties such as: software license, GPS usage was difficult as it covers small map scale and gives no exact coordination. To overcome these challenges, the system is designed to give instructions to the user by using APIs to get the current location of the user

7.3: Overall Strengths

User Interface Design:

The app's user interface design was generally praised for its intuitive navigation menus and layout. Elements such as clear labeling, visual cues, and logical grouping of content facilitated ease of use for many users, contributing to a positive overall impression of the app.

Familiarity and Consistency:

The app's interface design drew inspiration from popular mobile applications, incorporating familiar design patterns and conventions. This approach helped users feel more comfortable and confident in navigating the app, as they could leverage existing knowledge and expectations from other mobile experiences.

Immersive Multimedia Experiences:

One of the app's standout features was its integration of multimedia content, including high-resolution images. These multimedia assets enriched the visitor experience by providing additional context and historical background about the museum's artifacts.

Accessibility Enhancements:

Efforts to enhance accessibility for diverse user groups were positively received by users, who appreciated features such as adjustable font sizes, and language localization options. These accessibility enhancements ensured that the app was inclusive and accessible to users with varying needs and abilities.

7.4: Future Work

User-Centered Design:

To address usability issues, future iterations of the app will prioritize a user-centered design approach, involving end users in the design and testing process to ensure that their needs and preferences are effectively addressed.

Iterative Testing and Feedback:

Regular usability testing sessions will be conducted to gather feedback from users and identify areas for improvement. This iterative approach will allow us to refine the app's interface design, navigation pathways, and overall usability based on real-world usage patterns and user preferences.

Content Enhancement:

Collaborative Content Creation:

Collaboration with museum curators, historians, and subject matter experts will be essential to enhancing the quality and depth of content within the app. By leveraging expertise from diverse disciplines, we can ensure that new content is accurate, informative, and culturally relevant, meeting the educational needs of a broad audience.

Multimedia Enrichment:

Efforts will be made to enrich existing exhibits with additional multimedia assets, including high-quality images, videos, and interactive 3D models. By providing users with diverse and engaging content formats, we can enhance the immersive and educational value of the app.

Technical Optimization:**Performance Tuning:**

Technical optimizations will focus on improving the app's performance and stability, particularly when accessing multimedia content or interactive features. This may involve optimizing code, reducing resource-intensive processes, and implementing caching mechanisms to minimize loading times.

Expansion of Features:**Augmented Reality (AR) Experiences:**

The inclusion of augmented reality (AR) features was particularly well-received by users, who appreciated the opportunity to interact with virtual reconstructions of ancient artifacts and historical sites. AR technology added a new dimension to the museum experience, allowing users to explore virtual exhibits in a dynamic and engaging manner.

3D Models and Virtual Tours:

Interactive elements such as 3D models and virtual tours enabled users to explore museum exhibits in unprecedented detail. By offering immersive and interactive experiences, these features fostered a deeper understanding and appreciation of the museum's collection, especially for users unable to visit the museum in person.

Innovative Features:

Future versions of the Antiquities Museum app may introduce new features and functionalities to further enhance visitor engagement and educational outcomes. Ideas under consideration include real-time translation capabilities, personalized recommendations based on user preferences, and integration with social media platforms to facilitate sharing and community engagement.

Virtual Reality (VR) Integration:

Exploring the integration of virtual reality (VR) technology presents an exciting opportunity to offer immersive, lifelike experiences of ancient civilizations and archaeological sites. By leveraging VR hardware such as headsets and controllers, users can embark on virtual journeys through time and space, exploring historical environments in unprecedented detail.

Long-Term Sustainability:

Ensuring the long-term sustainability of the Antiquities Museum app will require ongoing support, maintenance, and funding. Strategic partnerships with governmental agencies, cultural organizations, and

7.5: Summary

In this final subsection, we provide a concise summary of the key points discussed throughout this chapter and the entire graduation project book. Our journey to develop the Antiquities Museum app has been marked by both successes and challenges, highlighting the complexities involved in leveraging technology to enhance cultural heritage preservation and education. Despite the identified weaknesses, the app has demonstrated significant strengths in user interface design, multimedia integration, and interactive features, laying a solid foundation for future development and innovation. By addressing usability issues, enhancing content quality, optimizing technical performance, and expanding features, we aim to ensure that the Antiquities Museum app continues to inspire and educate visitors for years to come, contributing to the mission of the Bibliotheca Alexandrina to promote knowledge and understanding of ancient civilizations.

References and Bibliography

- **Dart Programming Language Documentation:** <https://dart.dev/guides>
 - Official documentation for the Dart programming language.
- **Flutter Documentation:** <https://flutter.dev/docs>
 - Official documentation for the Flutter framework.
- **Geolocator Package Documentation:** <https://pub.dev/packages/geolocator>
 - Documentation for the Geolocator package, used for accessing device location.
- **Permission Handler Package Documentation:** https://pub.dev/packages/permission_handler
 - Documentation for the Permission Handler package, used for managing permissions in Flutter apps.
- **Firebase Documentation:** <https://firebase.google.com/docs>
 - Documentation for Firebase services, including Firebase Core, used for backend integration.

Bibliography

- Google LLC. "Firebase Documentation." Firebase. <https://firebase.google.com/docs>
- Google LLC. "Flutter Documentation." Flutter. <https://flutter.dev/docs>
- Flutter Community. "Geolocator Package." Pub.dev. <https://pub.dev/packages/geolocator>
- Baseflow BV. "Permission Handler Package." Pub.dev. https://pub.dev/packages/permission_handler
- Dart Team. "Dart Programming Language Documentation." Dart.dev. <https://dart.dev/guides>