浙江工业大学

本科毕业设计说明书（论文）

Undergraduate International Students' Graduation Project Report (Thesis)

Design and Implementation of an AI-Assisted Travel Planning App

学　　院：计算机科学与技术学院、软件学院

专　　业：软件工程（中外合作办学）

班　　级：2020 软件工程（中外合作办学）01

学　　号：202003340133

学生姓名：庄景文

指导老师：李小薪

提交日期：2024 年 6 月

浙江工业大学

本科生毕业设计(论文、创作)诚信承诺书

本人慎重承诺和声明：

1.本人在毕业设计（论文、创作）撰写过程中，严格遵守学校有关规定，恪守学术规范，所呈交的毕业设计（论文、创作）是在指导教师指导下独立完成的；

2.毕业设计（论文、创作）中无抄袭、剽窃或不正当引用他人学术观点、思想和学术成果，无虚构、篡改试验结果、统计资料、伪造数据和运算程序等情况；

3.若有违反学术纪律的行为，本人愿意承担一切责任，并接受学校按有关规定给予的处理。

摘　要

本文介绍了一款基于 OpenAI 的 ChatGPT 开发的 AI 辅助旅行规划应用程序，旨在提升个性化旅行体验。

自从第一个大型语言模型问世以来，其能力令人惊叹。研究人员和行业已经调查了其在众多应用中的潜力。ChatGPT 是市场上具有重要影响力的模型之一。同时，个性化旅行解决方案的需求增加和先进技术的需求为重新定义传统旅行规划范式提供了机会。但由于 ChatGPT 大部分情况下是纯文本处理模型，用户通过自然语言沟通得到的旅行信息是不够完善和直观的。通过将 ChatGPT 集成到 iOS 应用中，本项目探索了自动化和个性化旅行规划的新可能性，生成基于用户个人兴趣的旅行行程，包括一系列景点清单，景点清单中包括地图位置和描述等相关信息，用户也可以直接跳转至地图进行导航。此方法解决了通用旅行解决方案与个人偏好之间的差距，同时提供使用 AI 的便利性。

项目重点是设计和开发名为 iJourney 的 iOS 应用，该应用为用户提供一个使用 ChatGPT 生成个性化旅行行程的半自动化系统。应用本身按照行业标准设计，包括干净的架构和 Apple 的人机交互设计指南。该应用构建在 Apple 开发的最新框架之上，包括采用 MVVM 架构模式的 SwiftUI。iJourney 应用成功展示了如何利用 AI 提供动态和个性化的旅行规划解决方案。该项目展示了将 AI 与移动技术相结合的变革潜力，从而显著改善用户在旅行规划中的体验。

关键词： ChatGPT，大语言模型，iOS 开发，Swift 语言，MVVM 架构

ABSTRACT

This thesis presents the development of an AI-assisted travel planning app based on OpenAI's ChatGPT to enhance personalized travel experiences.

Ever since the first large language model came out, its capabilities were astonishing. Researchers and industries have investigated its potential in numerous applications. ChatGPT is one of the significant influential models on the market. Meanwhile, the increasing demand for personalized travel solutions and the need for advanced technologies offer an opportunity for redefining traditional travel planning paradigms. However, since ChatGPT is a plain text processing model in most cases, the travel information that users get through natural language communication is underdeveloped and intuitive. By integrating ChatGPT within an iOS app, this project explores new possibilities for automating and personalizing travel planning by generating itineraries that consist of a list of points of interest (POI) based on personal interests. The list of POI consists relative location and description information. The user is also able to jump straight into Maps to get direction. This approach addresses the gap between generic travel solutions and individual preferences, while providing the convenience of using AI.

The project focused on the design and development of the iOS app called iJourney, which provides users a semi-automated system for generating personalized travel itineraries using ChatGPT. The app itself was designed according to industry standards, including a clean architecture and human-computer design guidelines by Apple. The app was built on the newest frameworks developed by Apple, including SwiftUI, which adopts the MVVM architecture pattern. The iJourney app successfully demonstrates how AI can be harnessed to offer dynamic and personalized travel planning solutions. This project showcases the transformative potential of combining AI with mobile technology to significantly improve user experiences in travel planning.

Keywords: ChatGPT, Large Language Model, iOS Development, SwiftUI, MVVM

CONTENT

CHAPTER 1 INTRODUCTION

1.1 Background

In the realm of travel planning, the journey from conception to execution involves a very complex process of decisions and considerations. Historically, this process was usually done by travel agencies or self-held guidebooks, both of them are either not very affordable for economic travelers or lack of comprehensive information. However, with the development of internet and smart phones, the digital revolution led us enter the era of online platforms and mobile applications, transforming travel planning into a more accessible and customizable experience for individuals. While the digital tools bring convenience to travelers, the data explosion makes it to harder to make decisions. We often struggle to find a suitable trip for our own personal preferences in an ocean of travel recommendations efficiently. This is where the potential of artificial intelligence comes in. The recent heat of ChatGPT and other large language models (LLM) both in academia and industry demonstrated many novel solutions in many fields, including travel and tourism industry. The LLMs performs outstanding abilities in natural language process, sentiment analysis and recommendation. By utilizing LLMs, it is a great possibility to optimize the travel planning process by offering dynamic and personalized travel experiences efficiently.

1.2 Purpose

This thesis proposes the development of an AI-Assisted Travel Planning App - iJourney, designed specifically for iOS platforms. By integrating OpenAI's cutting-edge AI technologies ChatGPT, with Apple's development frameworks including SwiftUI and MapKit as well as the state-of-the-art programming language Swift, the project aims to explore a deep understanding of MVVM (Model-View-ViewModel) architecture pattern and develop a semi-automated approach to individual travel planning. This application will provide individual travelers a convenient way to explore potential travel destinations and generate personalized itineraries based on their interests and available time. The iJourney app will recommend cities to the user and create detailed itineraries for those cities once the user selects their desired destination

and travel dates, and the user can get location and description information easily, with just a tap, the user can get the directions to one of the points of interest. Through this project, we explore the collaboration between AI capabilities and mobile technology in creating a travel planning experience that is both intuitive and convenient.

1.3 Development Setup

The iJourney app will be deployed on the latest iOS platform (as the time of writing is iOS 17). The development environment will be set up on macOS Sonoma 14.5 with Xcode 15.4. This is the latest macOS and Xcode version as the time of writing. Using the newest Apple platform makes sure that the latest technologies can be utilized. Backwards compatibility will be a less concern since the iOS upgrade adoption rate is incredibly high.

The architecture of the application will be a best practice of MVVM pattern using SwiftUI (or clean architecture). The view (user interface) and model (business logic and data) will be separated by View Models which is a binder of the two. This ensures the decoupling of data and view and can improve the maintainability and testability of the application. SwiftUI and Combine are the two fundamental frameworks in this project. MapKit will be used to provide map related features.

The application will use HTTP requests directly to call ChatGPT's API and get response for the relevant data. This can eliminate the need of a server in this project so that we can focus on the iOS application itself. However, it would be recommended to have a dedicated server as a relay to have more control over the dataflow.

1.4 Document Structure

Chapter 1 delineates the foundational background and objectives of this project, along with the configuration of the development environment.

Chapter 2 examines the current state of research and the technologies pertinent to the system, and also does a competition analysis on the current market solutions.

Chapter 3 conducts a detailed analysis of the system designs, including functional and quality requirement specifications, architecture design and user interface design.

Chapter 4 discusses the implementation phase of the system, highlighting several pivotal technical aspects.

Chapter 5 provides a synthesis of the project and proposes a roadmap for ongoing and future research and development efforts.

1.5 Chapter Summary

This chapter mainly introduces the background, purpose of the project and briefly presents the development environment.

CHAPTER 2 RELATED WORK

2.1 Current Research Status

Here we did some literature review on some of the recent research papers to get an overall picture of the field.

2.1.1 ChatGPT and Its Application

Ever since the introduction of ChatGPT and other recent LLMs, researchers have been looking for their potential applications in main domains. The potential applications of ChatGPT and Large Language Models (LLMs) are vast and varied, reflecting their evolving capabilities across different fields.

According to Naveed et al. [1], LLMs like ChatGPT serve as foundational technologies in natural language processing, offering unprecedented opportunities in language translation, content creation, and automated customer service. Kaddour et al. [2] highlight the role of LLMs in enhancing machine learning tasks, including predictive text generation and sentiment analysis, which is required by this project's aim. Chen et al. [3] delve into the niche of prompt engineering, demonstrating how fine-tuning LLMs can unlock specialized applications in domains requiring nuanced understanding, such as legal advice or medical diagnosis. Fine-tuned LLMs can be a consideration in this project, but both the deployment of self-trained and fine-tuned open-source LLMs and fine-tuned GPT models provided by OpenAI would be overbudget, so we will mainly focus on stock GPT products which still has phenomenal performance. Wu et al. [4] underscore ChatGPT's significance in educational settings, suggesting its use as a tutoring aid, a tool for personalized learning, and a facilitator for research and writing. Nazir and Wang [5] and Gallardo Paredes et al. [6] extend the application scope to specialized fields like dentistry and software development, respectively, showcasing how ChatGPT can offer tailored information retrieval and integration solutions. These capabilities of LLMs could help with travelers to build their own trip plans based on their own personalities. Furthermore, Bahrini et al. [7] and Chowdhury and Haque [8] discuss the broader societal and economic benefits, including enhancing accessibility to information. Lastly, Fui-Hoon Nah et al. [9] envision a future where AI-human collaboration, facilitated by tools like ChatGPT, revolutionizes creative industries, research, and even daily personal interactions.

In terms of how we can use the LLMs from an end-user and application perspective, there are many works done regarding prompt engineering, which is a field of study that becomes popular after the explosion usage of ChatGPT. Even though some research showed that the training process of a language model would have more impact on result generating than well-designed prompts [10], [11], [12], using structured prompts can also create fine results. Arora et al. [13] developed a prompting method called AMA to improve the output result of LLMs and emphasize the importance of structured prompts for better results. Lu et al. [14] also introduced a new way that managed to improve the results of few-shot prompts. They have shown that well-designed prompts can substantially improve the result accuracy, so in this project, we would design structured prompts to achieve better results.

2.1.2 ChatGPT and Travel Industry

In the travel industry, the implementation of ChatGPT is poised to redefine customer service and travel planning. Gursoy et al. [15] provide an overview of current trends and future research directions, indicating how ChatGPT can transform hospitality and tourism through improved decision-making and customized service delivery. The potential of ChatGPT in enhancing information search and decision-making processes for travelers further underscores its significance in the travel industry [16]. Additionally, Demir & Sen Demir [17] suggest that ChatGPT can significantly impact service individualization and value co-creation in travel, highlighting its role in transforming industry practices for better consumer engagement and satisfaction.

2.1.3 Modern iOS Development

In the mobile application development area, there are a lot of new technology frameworks developed by people all around the world. People tried to build more advanced mobile applications with modern frameworks that usually required less effort for developers while achieving better performance with the current demand of the market. And with the advent of different operating systems and platforms, including iOS, Android, Web etc., developers tend to build their systems on all the platforms to serve the market in diversity, but this also comes with the cost of addtiontial development effort. Then it comes many different cross-platform frameworks that aims to lower the cost of multi-platform development of the same product. Some noteworthy frameworks like React-Native by Meta (formerly Facebook), Flutter by Google and Electron by OpenJS. This usually brings up the question of whether we should choose

to develop a product using cross-platform solutions or native platform solutions. The work by Kovács and Johanyák [18] shown that the performance of native iOS application is a big advantage against cross-platform solution which in this case Xamarin. Also the cross-platform solution often require more setup work to do before development. Even though there're not many research papers regarding this topic, many developer communities have a lot of discussion on this.

2.1.4 MVVM Architecture Pattern

In recent years, the MVVM (Model-View-ViewModel) architecture pattern has gotten more and more attention by the industry for UI application developments. Many web front-end framework and mobile application framework have adopted the MVVM architecture like Vue.js, React and SwiftUI. Ever since, there are several research being done to explore the difference between MVVM and MVC (Model-View-Controller), which is an older architecture pattern but still being used by many industries. The difference between the two is basically how would data interact with the view in the user interface. Different than MVC that using a controller to directly control the view, MVVM separates the business logic (Model) with the view (View) to reduce the coupling between the two, and it uses a binder to pass the values. Paramadani et al. [19] did a rendering performance comparison between MVVM and MVC architecture mobile applications and found out the MVVM exhibits superior UI rendering performance over MVC. A study done by Wilson et al. [20] also shows that the decoupling of MVVM improves the maintainability and testability of an application as well. Another study done by Magics-Verkman et al. [21] also points out that MVVM has better performance and better testability compared to MVC, while its easier to understand as well.

2.2 Related Techniques

2.2.1 ChatGPT

ChatGPT is a variant of the GPT (Generative Pretrained Transformer) language models developed by OpenAI, designed specifically for generating human-like text in a conversational context. It was first introduced by Brown et al. [22] in "Language Models are Few-Shot Learners". It leverages deep learning techniques to produce responses that mimic human conversation, making it particularly well-suited for applications requiring interaction with users in natural language. In your AI-assisted

travel planning app, ChatGPT could serve as the interface through which users communicate their travel preferences and receive personalized travel information. ChatGPT's ability to understand context and generate relevant, detailed responses can significantly enhance the user experience by providing tailored travel recommendations, answering queries, and facilitating a more interactive planning process.

2.2.2 iOS, SwiftUI and MVVM

iOS is the mobile operating system created and developed by Apple Inc. exclusively for its hardware. It is the software platform that runs on all iPhone models, providing the foundational interface and functionality users interact with. iOS is known for its intuitive design, security features, and robust performance. It offers developers access to a wide range of APIs and tools to create dynamic and engaging applications, ensuring a consistent and seamless user experience across all devices.

SwiftUI [23] is a user interface toolkit introduced by Apple in 2019 that allows developers to design and build graphical user interfaces across all of Apple's platforms, including iOS, macOS, watchOS, and tvOS. It uses a declarative syntax that makes it straightforward to design interfaces with less code than was traditionally required with Apple's older framework, UIKit. SwiftUI code automatically updates the view when the underlying data changes, thanks to its tight integration with the data-driven Combine framework. SwiftUI is naturally conducive to the MVVM architectural pattern.

MVVM (Model-View-ViewModel) architecture is a software design pattern that facilitates a clear separation of concerns in applications, particularly useful in graphical user interfaces. This pattern divides the application into three interconnected components: the Model, which represents the data and the business logic; the View, which is the user interface; and the ViewModel, which acts as a mediator by binding the Model data to the View. This separation not only simplifies the management of complex interfaces but also enhances the development process, allowing for more manageable code and improving the ability to test components independently of each other. MVVM is especially popular in applications built using platforms like WPF, Xamarin, or frameworks such as Angular, due to its strong support for data bindings and event-driven programming.

2.2.3 MapKit

MapKit [24] is a framework provided by Apple that allows developers to integrate

map-based services into their apps. It offers a rich set of features to display maps, annotate them with custom information, and implement location-based services directly within iOS apps. Developers can use MapKit to add interactive maps that can zoom in and out, show user location, and even provide turn-by-turn navigation. This makes it an invaluable tool for travel-related applications, where mapping and precise navigational instructions are crucial for enhancing user experiences.

2.3 Competition Analysis

Right now, there's actually not much application when it comes to travel itinerary generation using language models. With the advent of ChatGPT plugins and GPTs [25], many conventional travel all-in-one platforms have created their own GPTs that utilize both their APIs and ChatGPT's conversational interface. While these chatbots are useful if the traveler wants to talk about the itinerary with a bit of reference information, they are not meant to create a structural trip itinerary for the user.

2.3.1 GPTs

GPTs is a service OpenAI launched in November 2023. It is basically a way for other service providers to use the full power of ChatGPT and integrate their service into it. Each GPTs is like a few-shot models that is fed by several system prompts which are designed by third-party service providers and because of the learning capabilities of the models, they can do as the system prompts said to provide service according to the requirements.

KAYAK, as many people know, is a travel agency/consultant platform for people to search and decide their own travel itinerary. Before the appearance of GPTs, KAYAK already had their own website and services for manual and conventional travel searching. Now KAYAK also has built their own GPT on the platform provided by OpenAI. The KAYAK GPT is advertised to be the personal travel assistant for flights, hotels, and cars, which basically can let users ask questions in natural languages and the GPT is able to respond with natural languages as well, just like a human assistant.

The KAYAK GPT has received 3.8 stars score on the GPTs store and ranked number 5 in lifestyle (EN) section. This is a great tool for people who want to have somebody to chat about their travel plans, but it is also very limited. With only text information to be provided to the user with no map or geological information whatsoever, it is unclear for users where they're actually going.

2.3.2 Layla

This is a trip planner service that is both on web and app. It works very similar to the KAYAK on GPTs but with more functionalities with their partnership with SkyScanner and Booking.com. It is also a chat-based user interface which means users have to use natural languages to talk with the assistant. One thing that Layla does better than KAYAK GPT is that it can be run on an app which is more accessible for a broader user base (GPTs requires the ChatGPT Plus subscription). While this is a better form of trip planning, the way of using natural language still suffers from efficiency and the visualization of the plan.

2.4 Chapter Summary

This chapter explores the extensive applications of ChatGPT and other large language models in domains like customer service, medical advice, and travel planning. It highlights the shift towards native iOS development and the MVVM architecture for enhanced performance and maintainability in tech development. The chapter also notes the emerging use of ChatGPT in personalized travel itinerary generation, underscoring its transformative potential in various industries. At the end, the chapter did a competition analysis on two different trip planning services that are both powered by language models.

CHAPTER 3 SYSTEM DESIGN

3.1 Requirement Analysis

3.1.1 Overview

In this part, the requirements for the system are presented. The iJourney system is a complete app-based system that also has some third-party service access through HTTP. The analysis process is broken down to context, domain level requirements, product level requirements, data requirements and quality requirements.

3.1.2 System Context

To better analyze the structure of the whole system, a context diagram of the system was drawn, as shown in Figure 3-1. This diagram includes the user of the system, the system itself (iJourney) and ChatGPT as a third-party service.

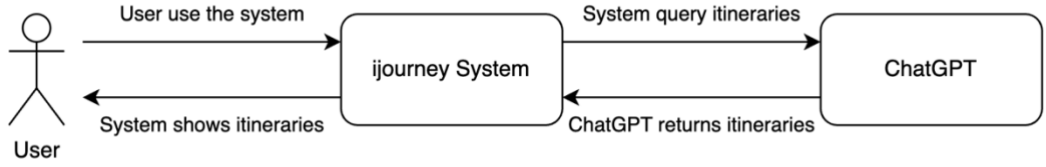

Figure 3-1 Context Diagram

- **User**

The target user should be those who are relatively familiar with iPhone or other mobile devices that uses a touch screen user interface. The only way a user interacts with the system is by using the app with a touch screen. The user will be able to adjust their interest levels, search for destinations and view itineraries through the system interface.

- **iJourney System**

This is the system that we are developing. The system provides user interfaces for the user to interact with and the system also has the ability to query ChatGPT for required information.

- **ChatGPT Service**

The ChatGPT service is provided by OpenAI through Microsoft's Azure cloud service. It is an API service so that the iJourney system can access it through HTTP requests.

3.1.3 Domain Level Requirements

- **Explore City**

  The user should be able to use related functionalities in the system to explore different cities to seek potential travel destinations.
- **Create and View Itinerary**

  The user should be able to use related functionalities in the system to create and view itineraries for different trips as their travel plans.
- **Manage User Profile**

  The user should be able to manage their profiles and other related information within the system.

3.1.4 Product Level Requirements

- **Explore City Domain**
  - The user should be able to see a list of featured cities in the city list page.
  - The user should be able to see each city's detail in the city detail page.
  - The user should be able to create a itinerary directly from the city detail page for this city.
- **Create and View Itinerary Domain**
  - The user should be able to see a list of already created itineraries in the itinerary list page.
  - The user should be able to enter the itinerary creation process by pressing the + button on the navigation bar of the itinerary list page.
  - The user should be able to choose the destination city and country of an itinerary during the creation process.
  - The user should be able to choose the starting date and ending date of the itinerary during the creation process.
  - The user should be able to adjust their interest level during the creation process.
  - The system should be able to aggregate all the data from the user and send them to ChatGPT in a consistent structure.
  - The system should be able to receive the itinerary generated by ChatGPT and store it into the data models of the app.
  - The system should be able to map all the points of interest using MapKit in Apple's map system.
  - The user should be able to see the details of each point of interest by choosing it on the screen.

- The user should be able to be taken to Apple's Map app to view even more detailed information if the user taps a button.
- The user should be able to delete the itinerary in the itinerary page.

➢ **Manage User Profile Domain**

- The user should be able to view their personal information in the profile page.
- The user should be able to adjust their interest levels in the profile page.
- The user should be able to adjust the general settings of the app in the profile page.

3.1.5 Data Requirements

The data in this system will mainly be divided into two parts: user and itinerary. All data would only be stored locally on device if necessary. All data are modeled in Swift and using Swift's type system. There will also be optional types which is a feature of Swift that allow us to use nil as a replacement if there's on data in this property.

When designing the data, we ensure that the data levels are the same as that used in JSON which will be used for communication between the application and ChatGPT. By doing so, we can easily parse the JSON data from the JSON using Codable protocol provided in Swift.

- **User Profile**

Table 3-1 User Data

Name	**Type**	**Optional**
id	UUID	/
name	String	/
birthday	Date	/
interests	InterestPreference	/

The user's data includes the basic of the user, as shown in Table 3-1, including username and his interest preference. The interests property is a type of InterestPreference which is defined in Table 3-2. Although the user's data is stored locally, there's possibility for future expansion that the data will be handled on the cloud, so the id property is reserved.

Table 3-2 Interest Preference

Name	Type	Optional
historical	InterestLevel	/
cultural	InterestLevel	/
nature	InterestLevel	/
entertainment	InterestLevel	/
shopping	InterestLevel	/
adventure	InterestLevel	/
relaxing	InterestLevel	/
food	InterestLevel	/
nightlife	InterestLevel	/

This type is used to store the user's interest level on all those different directions. Each of the interest is a type of InterestLevel, which is a enum type so that we can have non-continual segregated interest levels. This is because the interest level will be used in a natural language context, in which the ChatGPT will have a better understand of instead of using numbers.

Table 3-3 Interest Level

Name	Type	Optional
veryLow	enum raw	/
low	enum raw	/
medium	enum raw	/
high	enum raw	/
veryHigh	enum raw	/

The InterestLevel as shown in Table 3-3 has five different levels described in natural language. The natural language way of describing the level will help ChatGPT understand better since than using numbers.

- **Itinerary**

Table 3-4 Itinerary

Name	Type	Optional
id	UUID	/
destinationCity	String	/
destinationCountry	String	/
startDate	Date	/
length	Int	/
dailyItineraryList	[DailyItinerary]	/

As shown in Table 3-4, this is the top level of itinerary data, which stores the basic information like destination city name and destination country name. The property dailyItineraryList stores all the points of interest detail data about this itinerary.

Table 3-5 Daily Itinerary

Name	Type	Optional
id	UUID	/
poiList	[PointOfInterest]	Yes

DailyItinerary only consists of one property besides id which is the list of POI (points of interest) shown in Table 3-5. This data is solely for the segregation of POI list and the whole itinerary so we can have multi-day itinerary stored in a two-dimensional array.

Table 3-6 Point of Interest

Name	Type	Optional
id	UUID	/
name	String	/
type	InterestPreference	/
description	String	Yes
startTimeInDay	DateComponents	/
duration	DateComponents	/

As shown in Table 3-6, this is the bottom level data which stores each of the points

of interest. Each of the POI will have extensive information so that the user can easily understand where they're going, and this makes it easier to connect with MapKit. The user should be able to jump to Apple Maps right from each POI in the list to have more information.

3.1.6 Quality Requirements

- The system's user interface should be designed according to Apple's human interface guidelines that is consistent and simple to use and navigate.
- The system should be responsive for local tasks and reasonably responsive for remote tasks such as querying for the itinerary.
- The system should be able to insure the privacy of users' personal data and follow the corresponding regulations.

3.2 System Architecture

3.2.1 Overview

According to the requirements, the system needs to be responsive to users' actions. It is a reletively small system, and considering the main bottleneck for performance is only server side, which is the OpenAI service, so scalability is not a quality goal for the app development. The local performance should be on-part with other native iOS apps to provide a semeless user experience on the iOS platform. The usability of the system needs to be at the same level as the other native apps on the iOS platform as well. We need to make sure there's no major failure during the user experience to ensure the reliability of the system. The security would be a minor concern since the target user base is usually not senstive to basic personal data like birthday, and the app will only use local storage other than the ChatGPT service. The security of ChatGPT API will be covered by Azure.

The structure of architecture part of the document will follow the arc42's template. The main quality goals are listed in Table 3-7 and the main stakeholders are listed in Table 3-8.

Table 3-7 Quality Goals

Quality Goal	**Description**
Performance	The performance of the system should be on part with average iOS native apps that has great responsiveness.

Usability	The system should be easy to use for the user and does not require a long learning curve.
Reliability	The system should not have major failure during the time of use.

Table 3-8 Stakeholders

Stakeholder	**Description**
User	Currently this system will not have a business model, so the only stakeholder is the user.

3.2.2 Constraints

Architecture constraints are limitations or restrictions that shape the design and development of a system's architecture. Understanding and effectively managing architecture constraints is very important for a successful system design and implementation. Both technical constraints and other constraints are listed in Table 3-9 and Table 3-10.

Table 3-9 Technical Constraints

Constraint	**Description**
The system will only be developed on iOS platform.	Consider the scope of the project and the estimated effort taken, and that I am comparably more familiar with iOS development, we decided to only develop on iOS platform.
The system will only be developed using Swift programming language.	iOS apps can be developed both in Objective-C and Swift, but since I am more familiar with Swift, and Swift is a more modern and safer language, the system will only using Swift as the programming language.
The system will only be developed for the newest iOS version.	Considering the scope of the project, and that we want to use the newest frameworks from Apple, we can only develop the system by the newest standard. Since the adopt rate of iOS systems are very high compared to other systems, it is acceptable to drop the backward compatibility.

Table 3-10 Other Constraints

Constraint	Description
The ChatGPT API service will only be provided by Azure	Consider the budget of this project and the current cost of using ChatGPT API services, we choose to utilize the Azure's OpenAI API in a student account. The selection of model is limited by sufficient. GPT-4-0125 is the newest model available.
The development environment is limited to macOS.	To develop an iOS app and fully utilize all the Apple's framework, a macOS system is required.

3.2.3 Technological Solution Strategy

- **Programming Language and Frameworks**

For this project, because the target deployment platform is only iOS, so the programming language will be only chosen from either Objective-C or Swift. Objective-C is a relatively old programming language for Apple's platforms. It was a modified version of C with similar qualities of C. It is still being used by many legacy apps developed a few years ago, which proves its rigidity. But this project will start from starch, and Swift is a much more modern and safe language to use for less experienced developers, with its rich functionality sets and libraries like SwiftUI. Considering the performance quality goal with framed budget and development time, Swift is the best choice.

The fundamental framework for this project is SwiftUI. Compared to UIKit which is a legacy UI framework just like Objective-C compared to Swift, SwiftUI is simpler to develop and maintain with its MVVM architecture pattern. Other than SwiftUI, many other supporting libraries will be used as well such as MapKit which provides access to Apple's map data.

There are also other third-party open-source libraries being used to improve the UI build. Glur is a package that provides a gauss blur using Metal API.

- **Architecture Pattern**

In the system, we are using SwiftUI as the main UI framework. As such, MVVM is used as the main architecture pattern to determine the data flow and UI update cycles. There are two main data points that are used in the system, one is the itinerary and the

other is user's data. Both the data points combined will be the only data source for our system and the UI will only update when the data changes. This ensures the usability of our system by only providing the truth (the data source) to the users. The chance of having out-of-sync UI and data is reduced.

3.2.4 Building Blocks View

In this section, we present the building blocks view of the software system. The building blocks views highlight the key components of the system and their interrelationships. This provides a comprehensive understanding of the system's architecture and serve as a foundation for further discussions on design and implementation details. There are two different levels of the building blocks. Level 1 is the top level and illustrate the overall system architecture and relationships. Level 2 will be more specific about each module in level 1. The module in level 1 serves as a black box, but it will become a white box in level 2, which we can see what's inside the module.

- **Building Blocks Level 1**

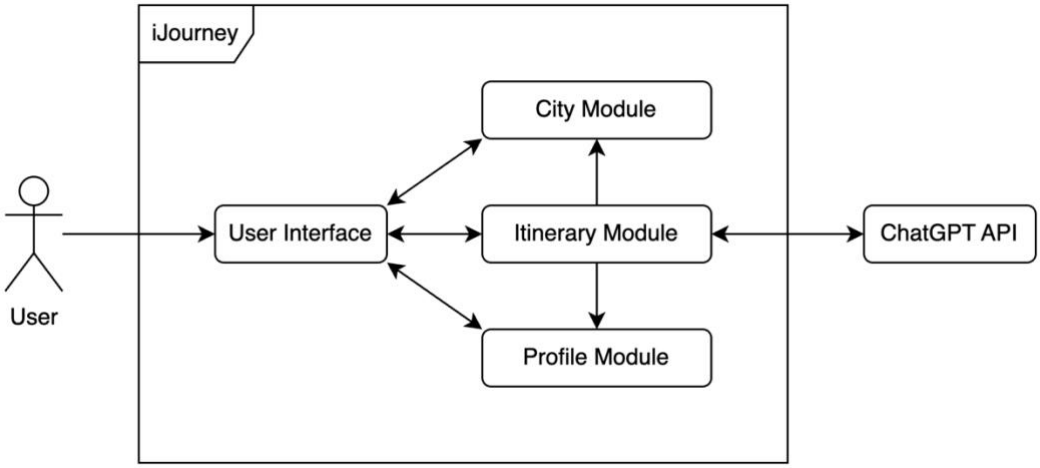

Figure 3-2 Building Blocks Level 1

As shown in Figure 3-2, in this level 1 building blocks view, we have three main parts within the system and two main actors outside the system. This view shows the very top level of the system. The Actor is the main user of the system, and he will only interact with the system through the user interface. Then the user interface will have interactions between both profile module and itinerary module. These two modules are separate because they serve different purposes and most of their data are separate. It is rather important in MVVM that we build the system according to the data flow and

location. The itinerary module will have access to the profile module to get relevant personal data of the user so that they can be used as part of the prompts for ChatGPT. Itinerary will interact with ChatGPT API through network calls. The itinerary modules will also have access to the model of the city module to access relevant city data.

- **Building Blocks Level 2 – City Module**

As shown in Figure 3-3, this is the level 2 building blocks view of the system, specifically the city module. It has two main parts within the module, city view model and city model. The city model stores the city and country's basic data like name and country code. Following the guidelines of MVVM, the city view model will have direct interaction to both the user interface and city model. The city model will have no direct relationships with the user interface. The city view model will pass along all the user intents and data updates from both sides for both sides. The itinerary module will also have access to the city model since it needs city data within the itinerary data.

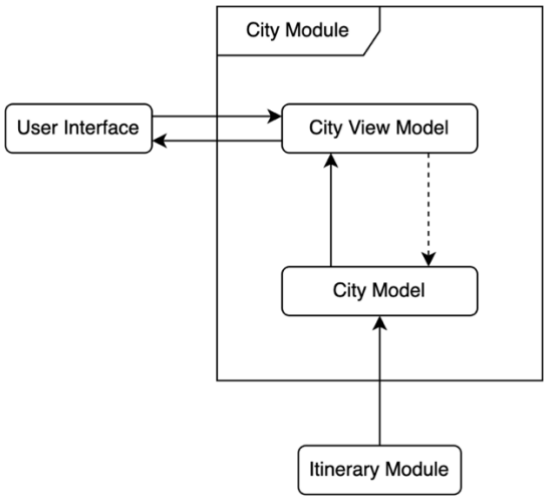

Figure 3-3 Building Blocks Level 2 - City Module

- **Building Blocks Level 2 – Itinerary Module**

As shown in Figure 3-4, this is the level 2 building blocks view of the system, specifically the itinerary module. It has three main parts within the module, which is more complicated than the profile module since it needs to deal with more data. The itinerary view model is basically the same functionality of any other view model. It is a bridge between the model and the user interface (or view). The itinerary model defines the data of each generated itinerary. The itinerary is too complicated to only have one model so we will have other supporting models that helps break down the structure also helps with JSON parsing etc. The itinerary view model will have the access to the

profile model when it needs to.

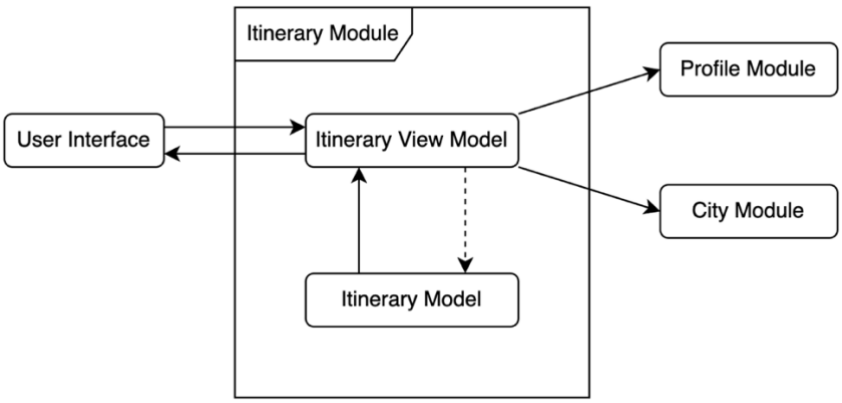

Figure 3-4 Building Blocks Level 2 - Itinerary Module

- **Building Blocks Level 2 – Profile Module**

As shown in Figure 3-5, this is the level 2 building blocks view of the system, specifically profile module. It has two main parts within the module, profile view model and profile model. The profile model is where all the user's personal related data is defined and stored. Following the guidelines of MVVM, the profile view model will have direct interaction to both the user interface and profile model. The profile model will have no direct relationships with the user interface. The profile view model will pass along all the user intents and data updates from both sides for both sides. The itinerary module will also have access to the profile model since it needs the personal data to generate prompts for ChatGPT.

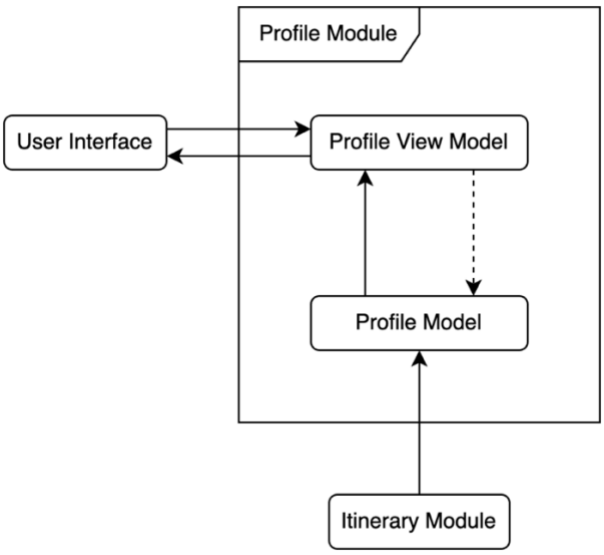

Figure 3-5 Building Blocks Level 2 - Profile Module

3.2.5 Runtime View

In this section, we present three sequence diagrams of the system's functionalities that is the most important. The sequence diagrams show how the user and system should do when executing a certain use case.

- **Create Itinerary**

As shown in Figure 3-6, this sequence diagram shows how a user can create an itinerary using the app. The user first will navigate to the itinerary page and interact with the user interface to put necessary information. And then the UI will call a function in the itinerary view model to execute a user intent. The view model will then pass the user intent to the model, at the mean time it will also request a copy of current user profile. And then the itinerary model will compose a prompt ready to send to ChatGPT API. Then the model will request itinerary generation by sending the completed prompt to ChatGPT API using network requests. Then the ChatGPT will process the request and return the results. After the data is updated in the itinerary model, the view model sees that and tells the UI to update as well.

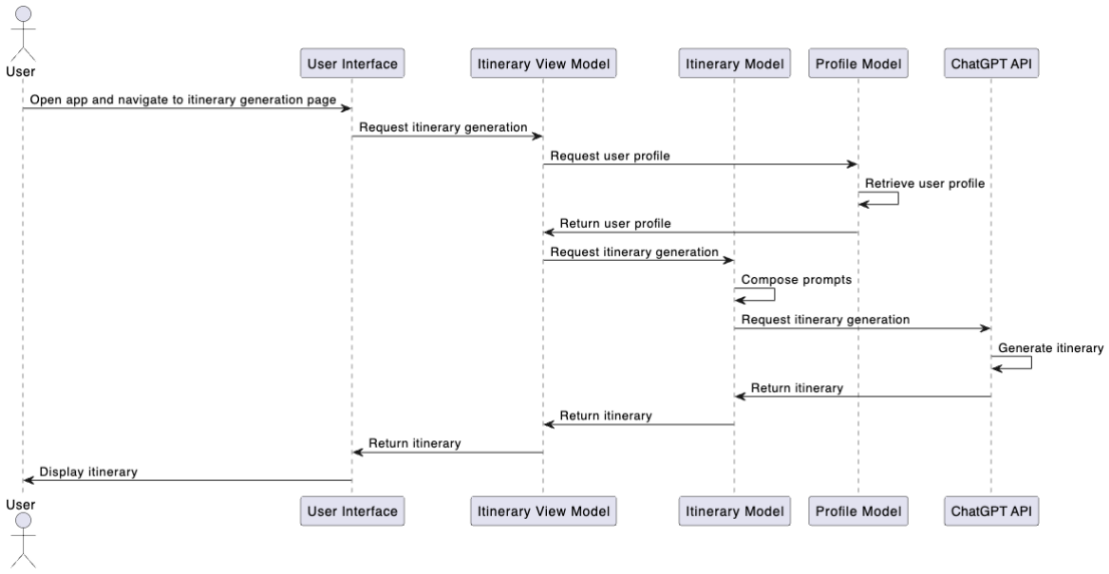

Figure 3-6 Create Itinerary

- **View Itinerary Details**

As shown in Figure 3-7, this sequence diagram shows how a user can view the detail of an itinerary that has been already generated. The user first needs to navigate to the detail page of the itinerary, and then the user interface will request for the specific detail data of the itinerary to the view model. Then the view model tells the model about

the user's intent. Then the model will return the related data so that the user interface can display them to the user.

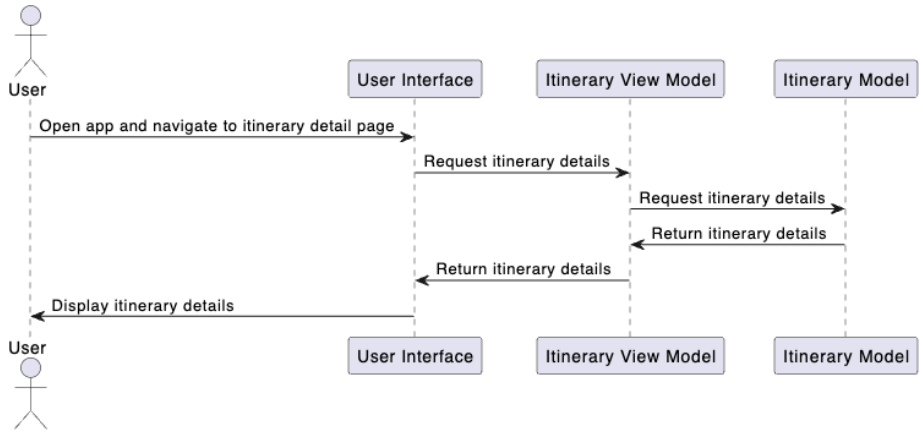

Figure 3-7 View Itinerary Details

- **Edit User Profile**

As shown in Figure 3-8, this sequence diagram shows how a user can edit their user profile. The user first will navigate to the profile page. The UI will request view model to retrieve the data and the view model will get the data from the model. Then the user can tap the edit button to edit the data, the view model will set the state to editing and temporarily save the edited data in memory. Then the user taps a button to save the data, the view model will then pass the new data to model and save it. Then the UI will update after the model changes the data source by notifying the view model.

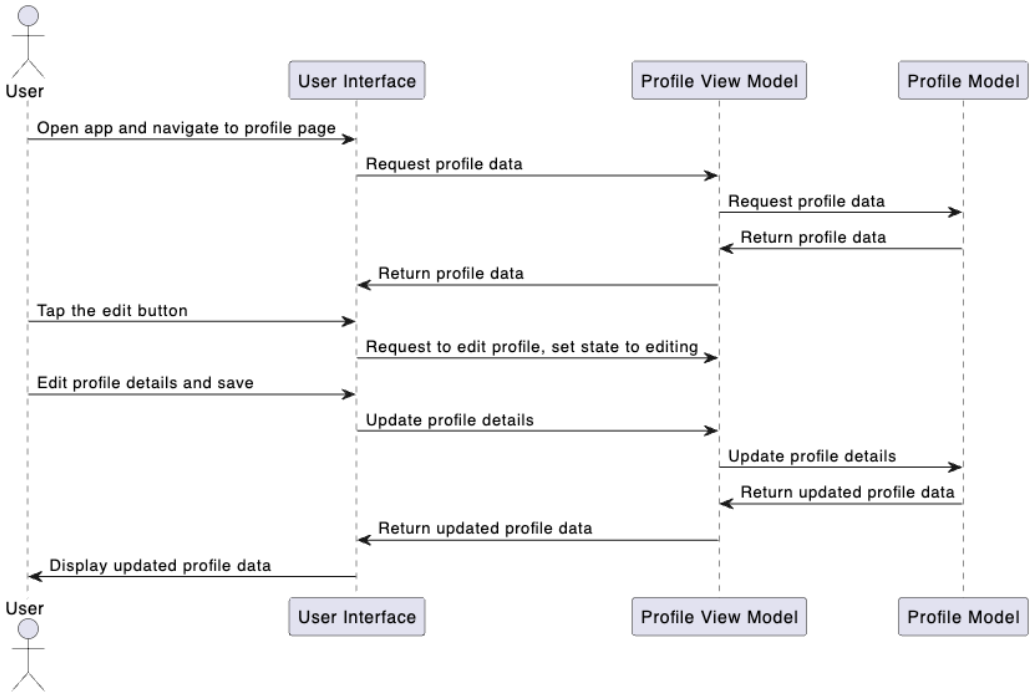

Figure 3-8 Edit User Profile

3.3 Prototype

3.3.1 Overview

In this section, we present the prototype design of the system that will help guide the development process as well as give a visual impression on the system. The majority of the design is done by following the Human Interface Guidelines by Apple. The Human Interface Guidelines (HIG) from Apple are a comprehensive set of recommendations and best practices designed to help developers and designers create intuitive, consistent, and user-friendly interfaces for Apple platforms, including iOS, iPadOS, macOS, watchOS, and tvOS. These guidelines cover a wide range of topics to ensure a cohesive user experience across all Apple devices, which in this case, is an iOS device.

Because we are using SwiftUI as our main development framework, and SwiftUI is a very intuitive UI framework that allows the developers to preview the layout of all the UI when typing the code, which tremendously help speed up the prototype phase. As we are taking that as an advantage, we will use SwiftUI for all the prototype design, not only because of its tight connection between development and design, but also it strictly makes us to follow the guidelines of HIG.

The system supports both light mode and dark mode using iOS system's settings, as shown in Figure 3-9, which follows the best practices according to the HIG. The system uses green as the tint color which separate itself from the default app of iOS system, while maintain the readability and usability both in light mode and dark mode.

Figure 3-9 Light and Dark Mode

3.3.2 Navigation Hierarchy

This system is not trivial so some kind of navigation will be needed for the user to reach all the functionalities of the system. In this system, three main ways of navigation will be used: tab, navigation push and modal sheet. These three navigation methods are all very common on iOS device and are recommended by HIG. The clear navigation can bring a clear map of operation to the user, so they won't get lost when using the system.

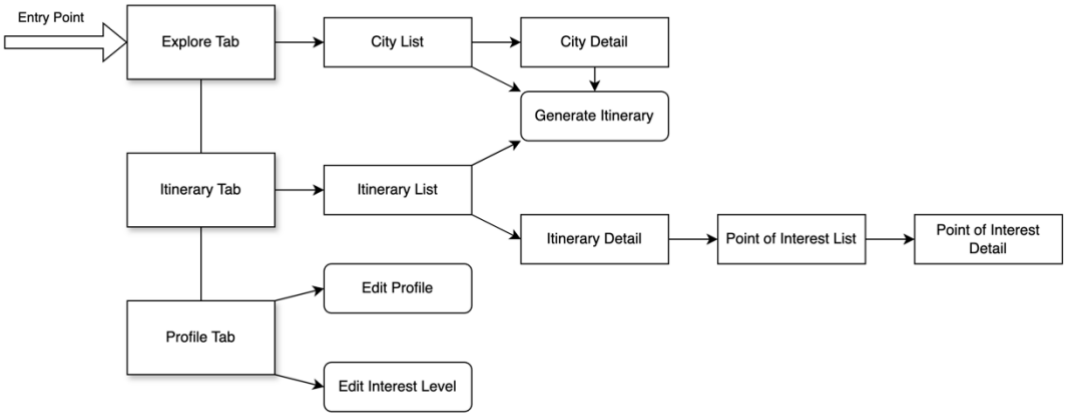

Figure 3-10 Navigation Flow

As shown in Figure 3-10, this is the navigation flow of the system. The main entry point is where the user will be shown when launching the app. All tab views are marked using big rectangular boxes with shadows. All navigation destinations are all marked with regular rectangular boxes. The rounded rectangular boxes are modal sheets which will pop up from the bottom of the screen if called. The itinerary tab and the profile tab are parallel to the explore tab, where the user can navigate to using the tab bar at the bottom of the screen. The explore tab shows a list of recommended cities, which can be further navigate down to the detail view. The itinerary tab shows a list of already created itineraries which can also be navigate down to view the detail, which includes lists of point of interests in each day. The profile tab shows all relevant user's information and two navigation destinations for the user to edit the information. The generate itinerary sheet can be access both from the itinerary tab with a toolbar button or directly from the city's views.

3.3.3 Explore Tab and City Detail

The explore tab is the starting point of the system, as shown in Figure 3-11 (left). The system will show a list of recommended cities as potential destinations for the user. The list is made of cards, each card has an image of the destination city with its name. The cards' background has a gradient color that are sampled from the image.

The user can tap the card and the system will navigate into a city detail page where there will be an image of the city, a map shows where the city locates and a text description about the city so the user can know more detailed information, as shown in Figure 3-11 (right). There are also menu buttons made of ellipsis where the user can tap, and the system will bring up a context menu where the user can choose to start an itinerary with the selected city. There is also a button right above the map that will take the user to the Maps app to see nearby places.

Figure 3-11 Explore Tab (Left) and City Detail (Right)

3.3.4 Itinerary Tab and Itinerary List

As shown in Figure 3-12 (left), this is the itinerary tab where everything related to itinerary is located here. The system will show a list of itineraries that has been already generated by the user. The list is made of cards, just like the city list. Each card will have an image of the destination city as background with the name of the city and time on top of it.

When the user taps the card, the system will navigate to an itinerary detail page where the user can view all the details of the itinerary information as shown in Figure 3-12 (right). There are mainly three sections in the detail page, the first one is the image of the city at the top with its name and dates of the itinerary. The second section is a map view which will display where the city is on the map. The third section is a list of point of interests for each day in the itinerary.

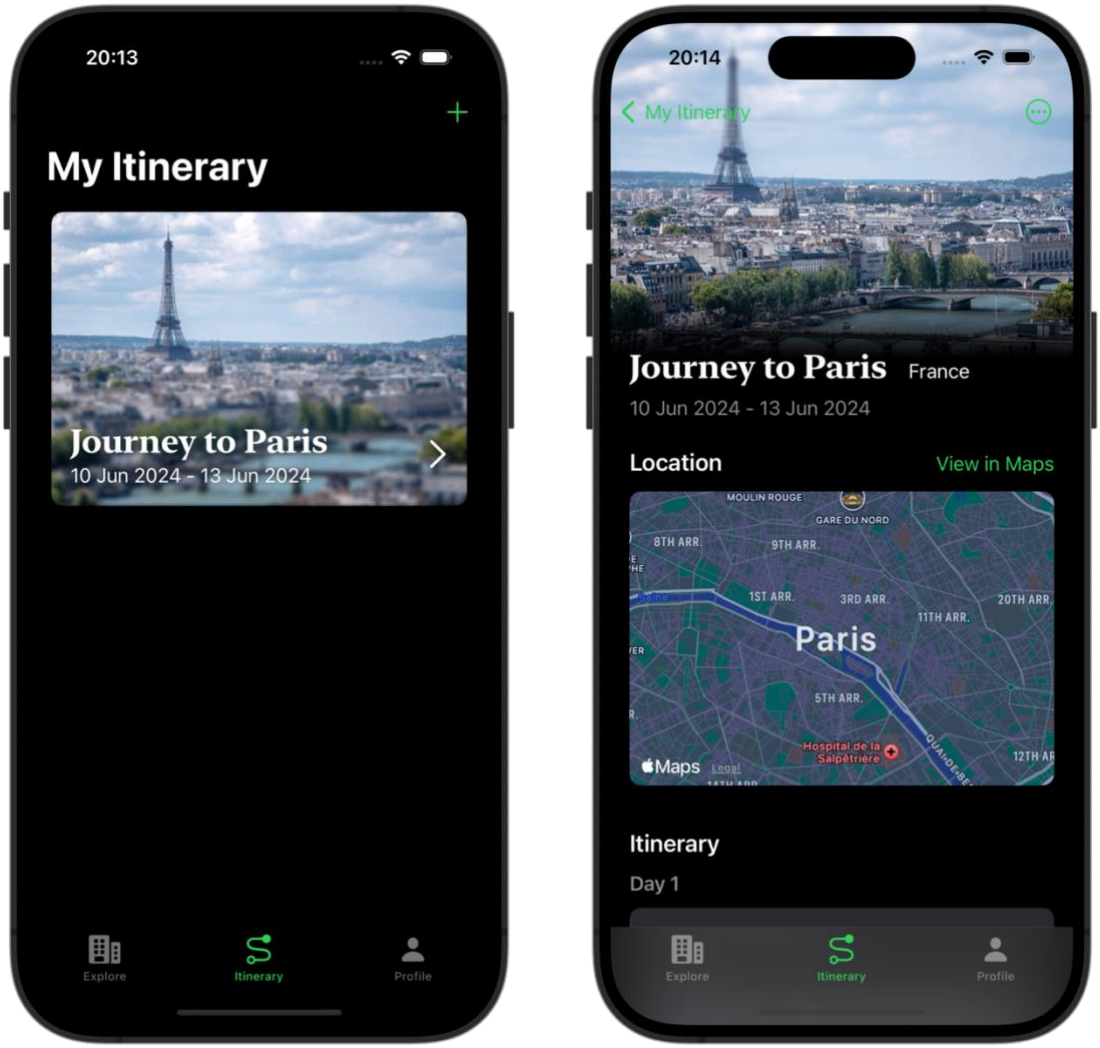

Figure 3-12 Itinerary Tab (Left) and Itinerary List (Right)

3.3.5 Point of Interest List and Detail

Scrolling down the detail page of each itinerary, in the third section of the itinerary detail page as shown in Figure 3-13 (left), there will be lists of point of interest (POI). Each list will be separated by a marker of day number, indicating which day would this list of POI be. In each list, there will be cards of point of interest which has the name of the POI, type of the POI and an image of the POI.

The user can tap on the card and the system will navigate to a POI detail page as shown in Figure 3-13 (right). The detail page of POI is similar to the city detail page, where there will be an image, a map, and a description of the POI.

Figure 3-13 POI List (Left) and POI Detail (Right)

Inside the detail page of each point of interest, there will be a map view appearing if the system is able to search this place using MapKit. The map will pinpoint the location of the point of interest so that the user can have an idea about what's around this location. There will also be a button labeled "View in Maps", where when the user taps it, the system to automatically jump to Apple's Maps app, and navigate to this specific location, so that the user can have even more information to view on, as well as getting directions directly in the Maps app. The process is shown in Figure 3-14.

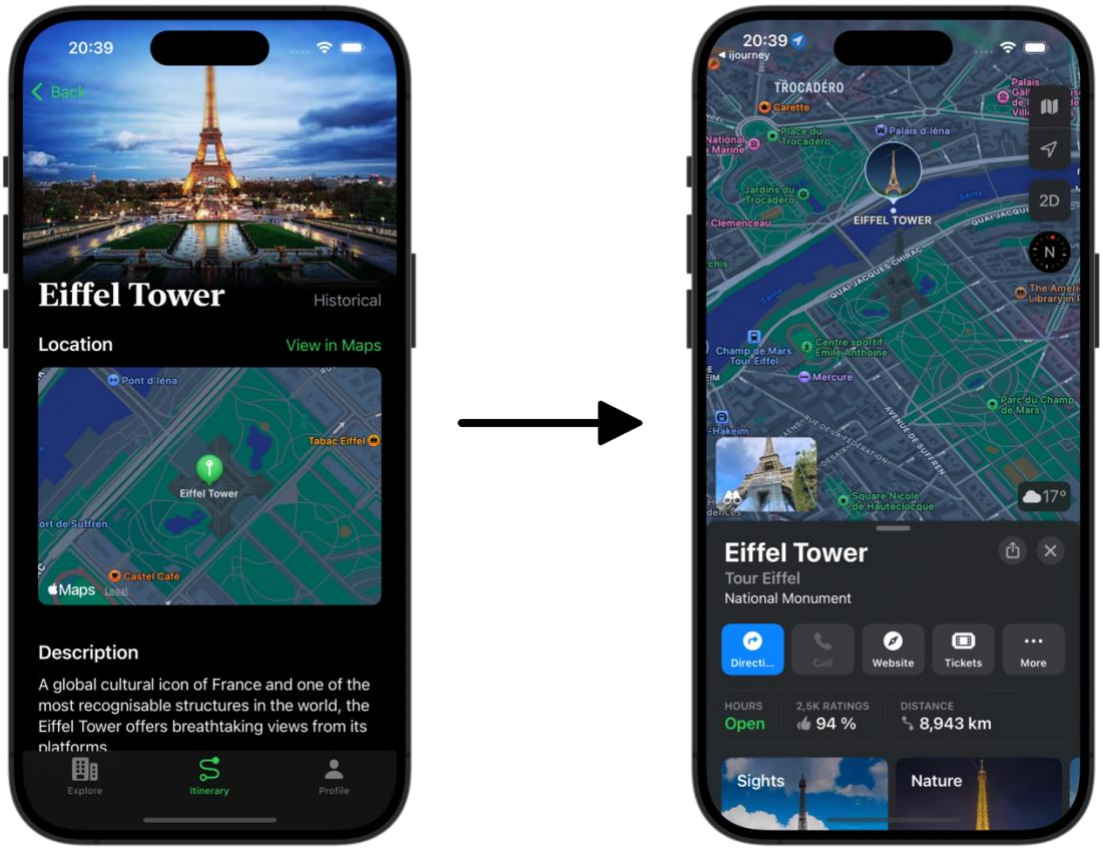

Figure 3-14 Jump to Maps

3.3.6 Generate Itinerary Modal

As shown in Figure 3-15, when the user taps the "plus" button on the navigation bar in the itinerary list page, a modal sheet will pop up from the bottom of the screen. On the modal there will be a list of country for the user to choose. After the user choose a country, there will be a city list of the selected country. Both lists are searchable so that the user can find the desired city more easily by just typing the letters in the search box.

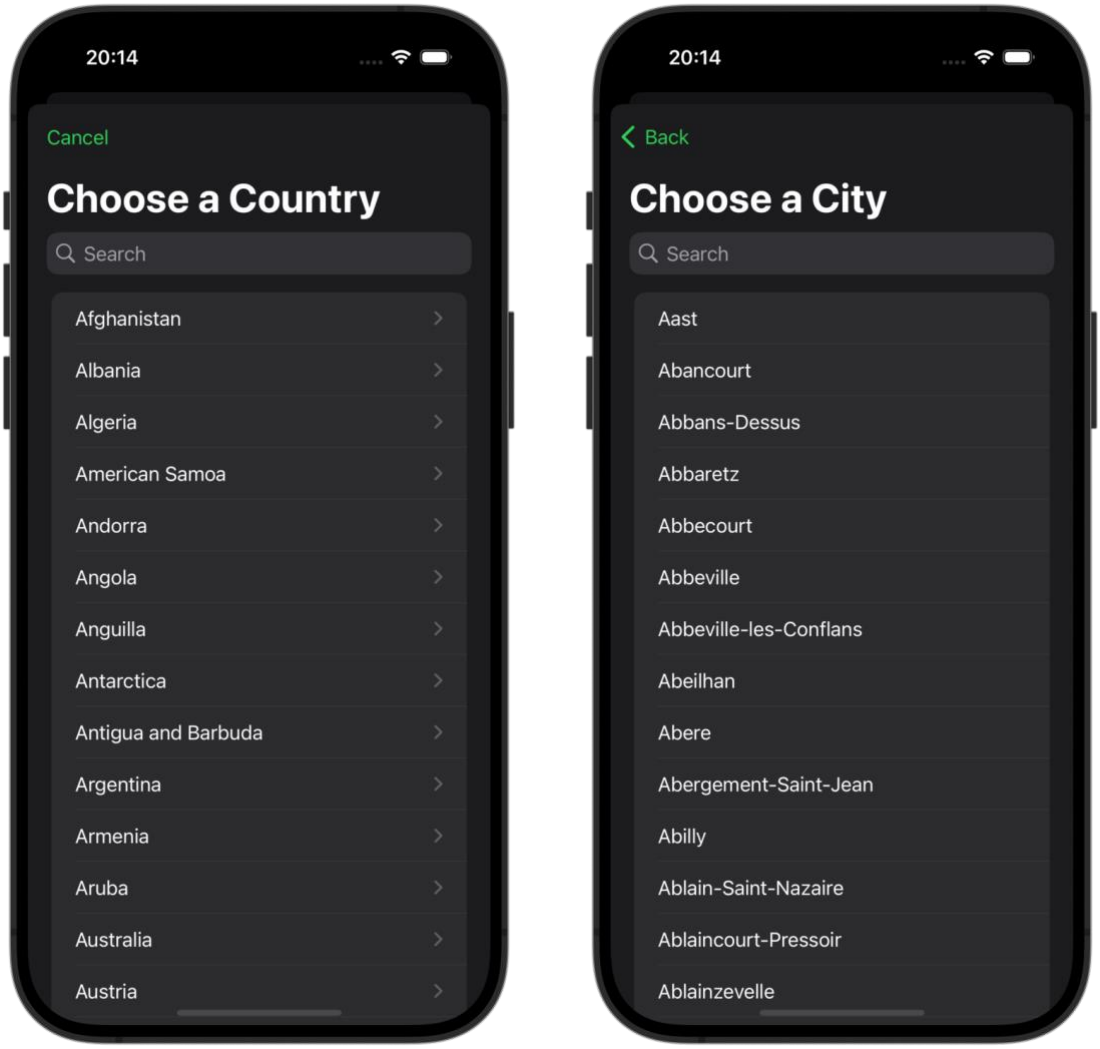

Figure 3-15 Choose a Country (Left) and Choose a City (Right)

After selecting the city, another full screen cover (a modal that covers the whole screen) will pop up on top of the current modal as shown in Figure 3-16 (left). This is the generate itinerary page where the user would confirm relevant information about the desired itinerary and make adjustments before sending it to ChatGPT. The user will see an image of the selected city, the name of the city and country, and the user can choose the start and end date of this itinerary.

The user can also adjust their interest level by tapping the Edit Interest Level button where another modal will pop up as shown in Figure 3-16 (right). The edit interest level modal is for the user to adjust their interest in each category so the ChatGPT can take that as a reference when generating the itinerary. After confirming all the information, the user can tap the create button on the top right side of the screen to send the request. This generate itinerary page can also be accessed from the city list or detail. So that

when the user thinks the city in the recommendation list is their choice, they can create an itinerary right away.

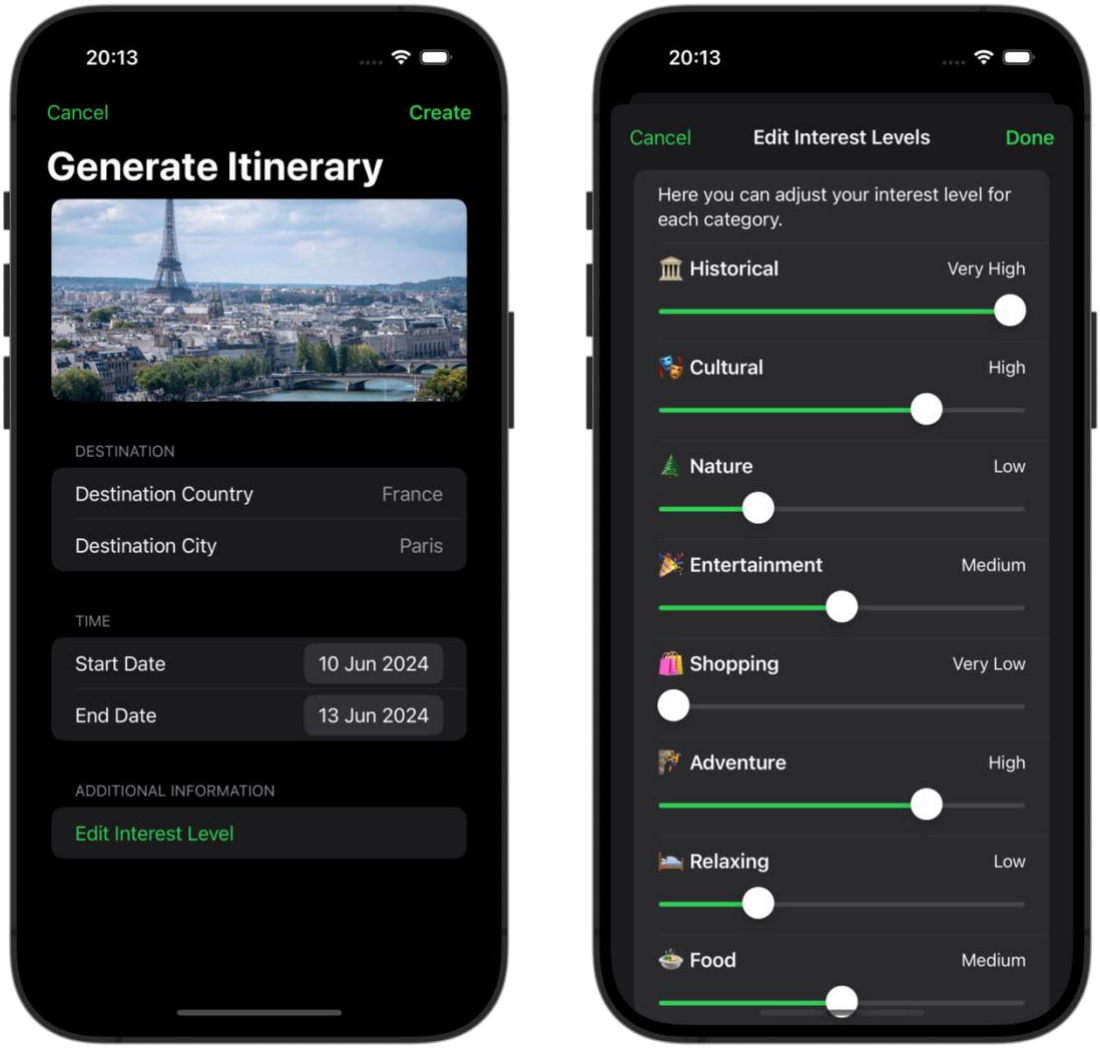

Figure 3-16 Generate Itinerary (Left) and Edit Interest Levels (Right)

3.3.7 Profile Tab

As shown in Figure 3-17 (left), this is the profile tab. The user is able to view their personal information, which includes the name, birthday and language preferences. There are two buttons in the second section where the user can edit their profile or edit their interest level, just like in the generate itinerary page.

When the user taps the edit profile button, a modal sheet will pop up where the user can edit the information, as shown in Figure 3-17 (right). The user cannot dismiss the modal interactively if they have already changed something to prevent accidental

operations. When the user taps the cancel if they have made some changes, there will be a confirmation dialogue pop up from the bottom of the screen that gives the user two choices of discard changes or keep editing.

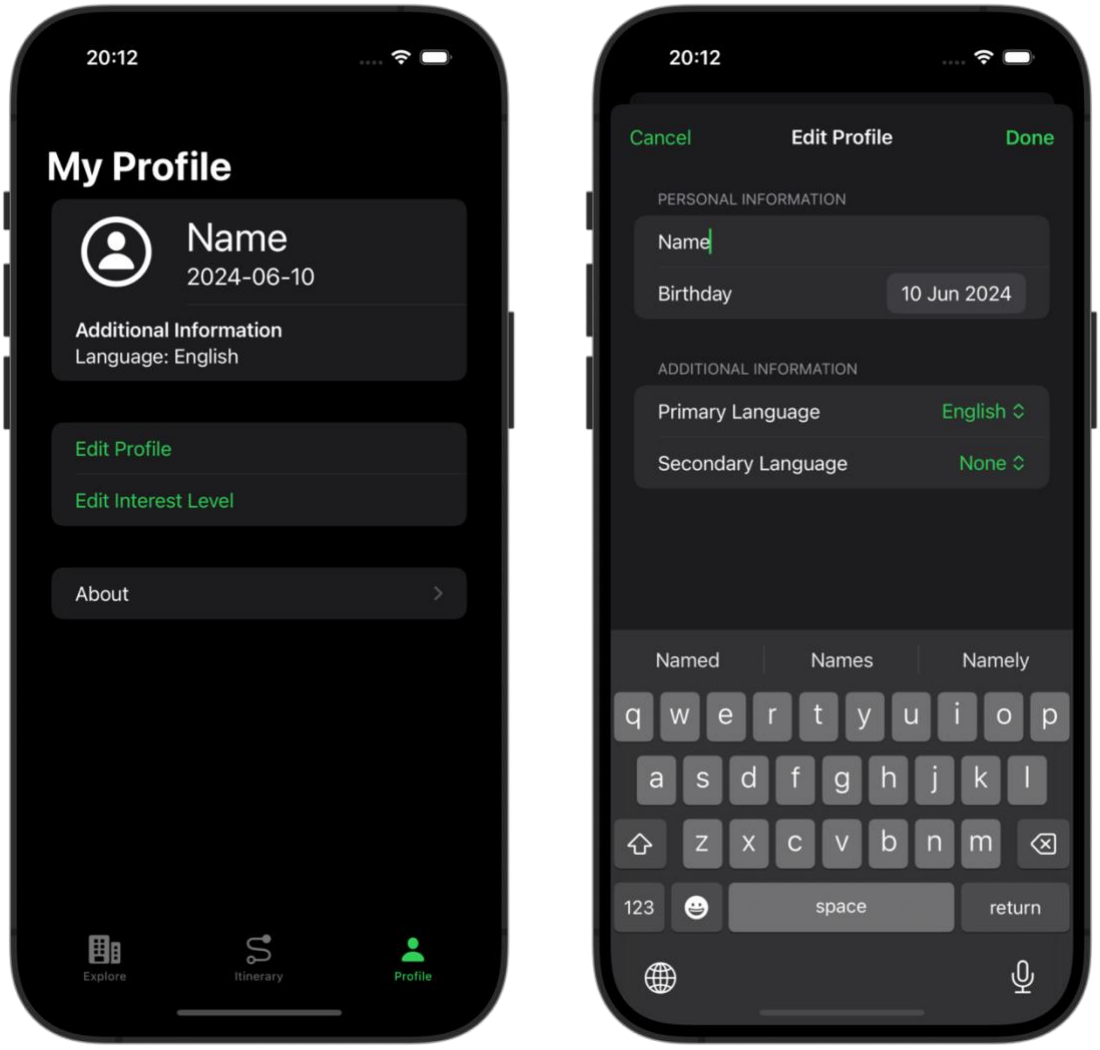

Figure 3-17 Profile Tab (Left) and Edit Profile (Right)

3.4 Chapter Summary

This chapter delves into the comprehensive design of the AI-assisted travel planning app, iJourney. It begins with a detailed requirement analysis, outlining the system's context, domain, product, data, and quality requirements. The discussion then presents the system architecture, including the technological solution strategy and building blocks view, highlighting key components and their interactions. The runtime

view illustrates the execution of core use cases, while the prototype section offers a visual representation of the user interface and navigation hierarchy, adhering to Apple's Human Interface Guidelines. The chapter concludes with an overview of the system's design, setting the stage for the implementation phase.

CHAPTER 4 SYSTEM IMPLEMENTATION

4.1 Development Environment Setup Details

The development environment is crucial to the smooth implementation of the system. We have chosen the newest development tools to help us achieve the goal. Here is a comprehensive list of the environment.

- Hardware: MacBook Pro 14" 2021
- Operating System: macOS Sonoma 14.5

- IDE: Xcode 15.4.0
- Version Control Client: Tower
- Version Control Server: GitHub
- Network Tool: Proxyman

- Main Programing Language: Swift
- Main UI Framework: SwiftUI
- Other Frameworks: Foundation, Combine, MapKit, XCTest
- ChatGPT API Version: GPT-4-Turbo-04-09 (2024)

These tools are modern and up to date. Everything is running on the stable version. Xcode is the only IDE we will use since it provides the most tools directly comes from Apple. Tower will be used as the local version control software, paired with GitHub as the remote repository. Version control can provide additional backups to the code base while keeping track of the progress. Proxyman is the network tool since we need to diagnostic network connections when doing test runs.

4.2 Key Technologies and Implementation

In this section, we will discuss about different key technologies and decisions that are influential to the system. These technology details are mostly code-level and they are the most fundamental and important part of the system.

4.2.1 Data Model Structure

The models are designed followed by the data requirements in Chapter 3 with additional supporting models. All the models can be grouped by three main parts: city related, itinerary related and profile related.

- **City Group**

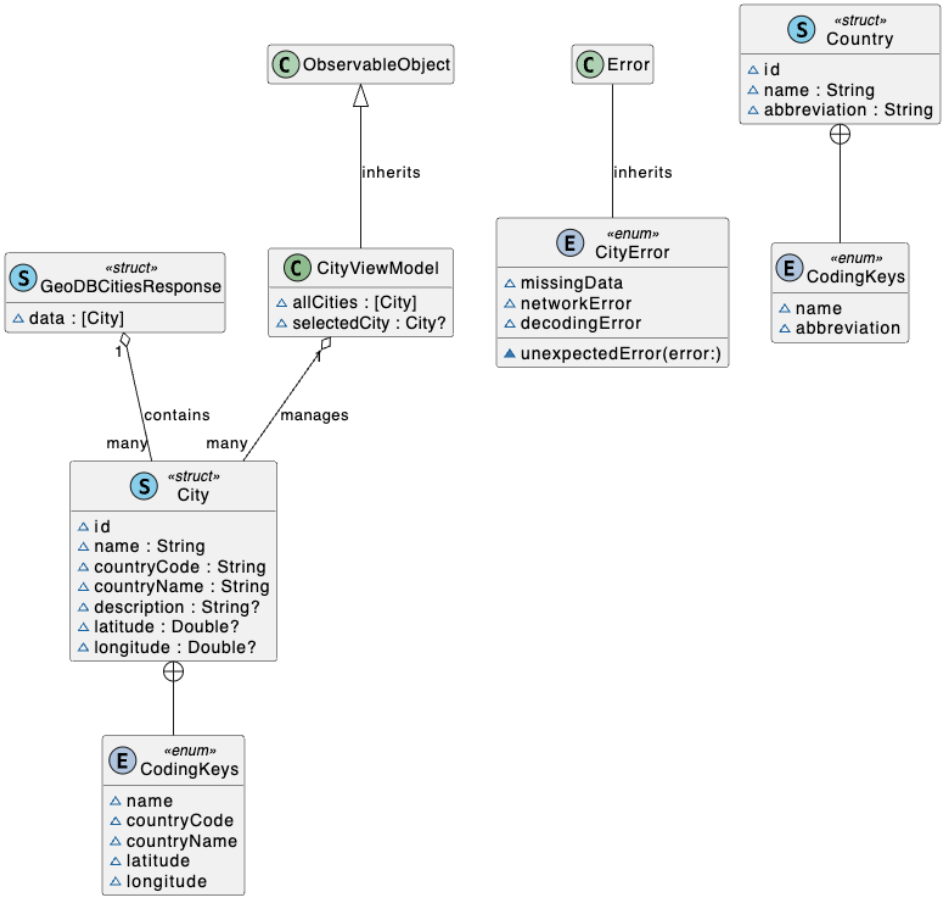

Figure 4-1 City Group Class Diagram

As shown in Figure 4-1, this is the UML diagram represents a set of models dealing with city and country data in our system. The primary models are City and Country. GeoDBCitiesResponse is only used when retriving data from the GeoDB API. There are CodingKeys defined within City and Country, which are both used for JSON coding. These models are interconnected to manage and represent city and country information within the app, with the view model to connect with the views. CityError is mainly used for handling networking errors. Notice that the Country and City are not directly related, that's because of the JSON structure of the GeoDB API is only using countryName as the property, adding the country struct to the city will make the coding process more complicated.

- **Itinerary Group**

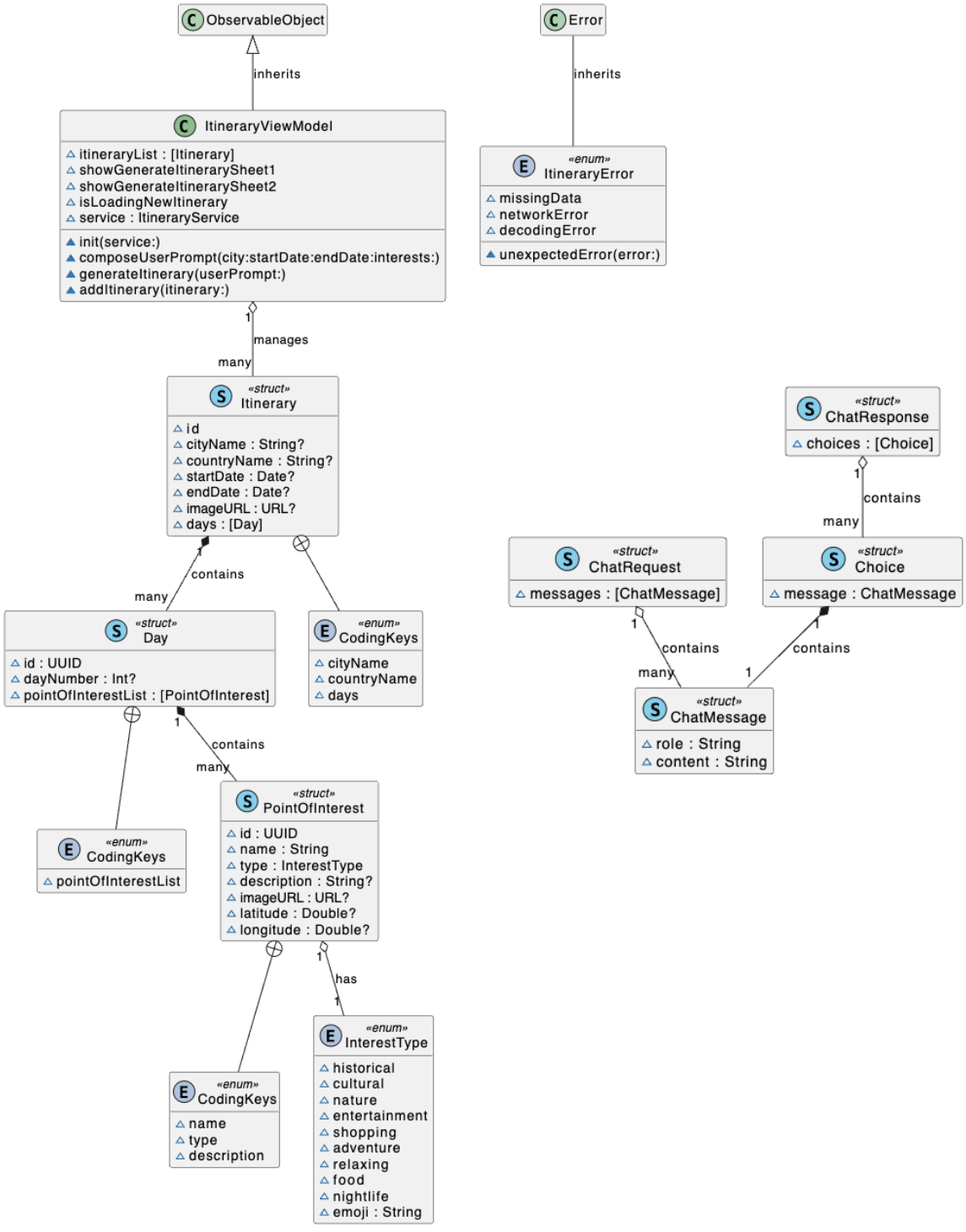

Figure 4-2 Itinerary Group Class Diagram

As shown in Figure 4-2, this is the UML diagram that represent the models dealing with Itinerary. This is quite a big group since it needs to handle more information. The primary models are Itinerary, Day and PointOfInterest, which they combined to be the core model of the system. In one itinerary there will be multiple days, in one day there

will be multiple POIs. Under each of the primary models, there is a CodingKeys enum that will help the decoder to decode JSON structure into the model when we receive the itinerary from ChatGPT. The view model is used to manage the data flow between model and the view as well as the states. There is a separate subgroup of models inside this group which is responsible for coding the ChatGPT API requests and responses.

- **Profile Group**

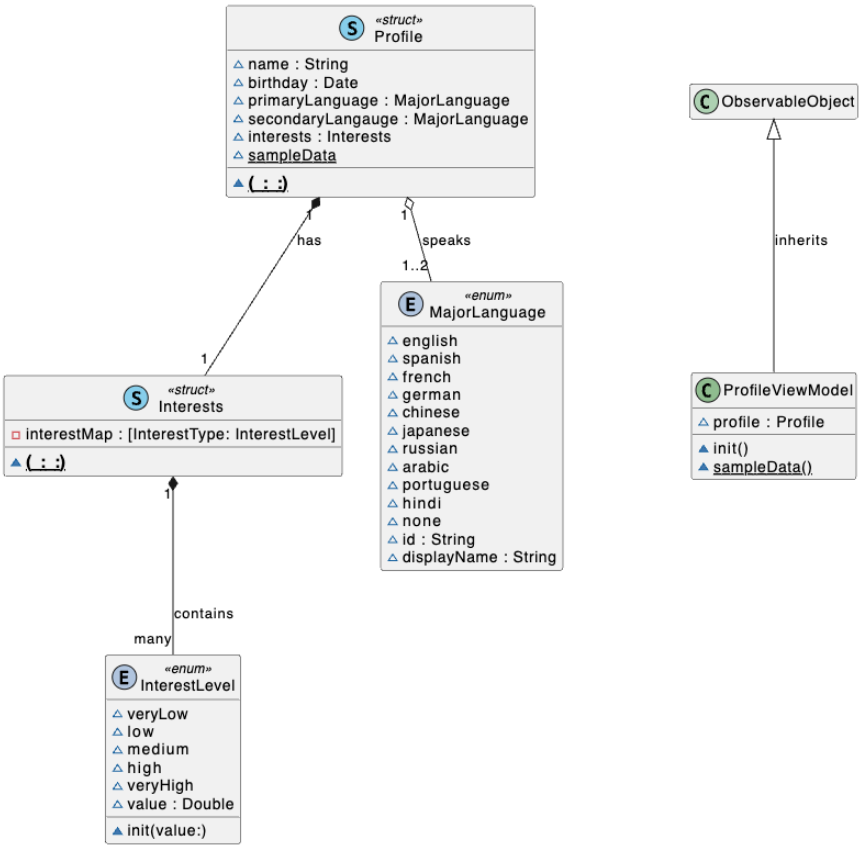

Figure 4-3 Profile Group Class Diagram

As shown in Figure 4-3, this is the UML diagram of the profile group, which is responsible for managing the user's profile information. The basic information is represented directly inside the Profile struct, and the interest levels are represented by an Interests type which has a dictionary of two enumerations inside. The InterestType is defined in the itinerary group but will also be used here. The MajorLanguage is an enumeration that stores a limited number of languages hard-coded.

4.2.2 JSON Parsing and Mapping in Swift

Different than JavaScript or Python, Swift is a strong-type programming language. This means Swift is safer and has better performance, but it also means that when it

comes to JSON parsing, it will need more steps before we can use the data in the memory. In our system, we need to decode a JSON response from network requests twice to get the information we desired, since the information JSON is generated by ChatGPT while the ChatGPT API itself is using JSON as response.

```
{
  "city_name": "<CityName>",
  "country_name": "<CountryName>",
  "itinerary_list": [
    {
      "poi_list": [
        {
          "name": "<Name>",
          "type": "<InterestType>",
          "description": "<Description>"
        }
      ]
    }
  ]
}
```

Figure 4-4 JSON Template for Itinerary

```
struct Itinerary: Identifiable, Codable {

  let id = UUID()
  var cityName: String? = nil
  var countryName: String? = nil
  var startDate: Date? = nil
  var endDate: Date? = nil
  var imageURL: URL? = nil
  var days: [Day] = []

  enum CodingKeys: String, CodingKey {
    case cityName = "city_name"
    case countryName = "country_name"
    case days = "itinerary_list"
  }

  init() { }

  init(from decoder: any Decoder) throws {
    let container = try decoder.container(keyedBy: CodingKeys.self)
    self.cityName = try container.decode(String.self, forKey: .cityName)
    self.countryName = try container.decode(String.self, forKey: .countryName)
    self.days = try container.decode([Day].self, forKey: .days)
  }
}
```

Figure 4-5 Part of Itinerary Model

As seen in Figure 4-5, this is part of the Itinerary struct that needs to be decoded from a JSON generated by ChatGPT where the JSON structure is pre-defined in the prompt. Figure 4-4 shows the template structure that should be generated by ChatGPT

and should be decoded into an itinerary. Inside the itinerary struct, beside the id and days property, all other properties are optional since we will want to have the ability to create an empty itinerary in some case. The obvious problem here is that the structure of itinerary model is not a one-to-one map of the template JSON and even the names are different because of different code styles (JSON uses underline while Swift uses camel). To address this problem, we first need to make the itinerary model codable with the Codable protocol. This will allow the Swift to automatically includes a initializer to decode the JSON to the model. Then we need to define a set of CodingKeys because of the naming differences. The CodingKeys is an enumeration inside the model, where each case inside the enumeration is a key in the JSON that needs to be decoded. Each cases' name should be the same as the property that needs to be mapped to. And we are also using String protocol so that we can eliminate the naming differences by assigning a String value to each case. After setting up the CodingKeys, we should also include a custom initializer since the one comes with the Codable protocol will no longer work because it doesn't know which key to decode to. Then in the initializer we can get the JSON container first and then assign each value to the property using the right key.


```
{
    "id": "chatcmpl-6v7mkQj980V1yBec6ETrKPRqFjNw9",
    "object": "chat.completion",
    "created": 1679072642,
    "model": "gpt-35-turbo",
    "usage":
    {
        "prompt_tokens": 58,
        "completion_tokens": 68,
        "total_tokens": 126
    },
    "choices":
    [
        {
            "message":
            {
                "role": "assistant",
                "content": "Yes, other Azure AI services also support customer managed keys. Azure AI services offer multiple options for customers to manage keys, such as using Azure Key Vault, customer-managed keys in Azure Key Vault or customer-managed keys through Azure Storage service. This helps customers ensure that their data is secure and access to their services is controlled."
            },
            "finish_reason": "stop",
            "index": 0
        }
    ]
}
```


Figure 4-6 ChatGPT API Response Template

```swift
struct ChatMessage: Codable {
  let role: String
  let content: String
}

struct ChatRequest: Codable {
  let messages: [ChatMessage]
}

struct ChatResponse: Codable {
  struct Choice: Codable {
    let message: ChatMessage
  }
  let choices: [Choice]
}
```

Figure 4-7 ChatGPT API Response Model

In Figure 4-6, this is a response sample from Azure's OpenAI REST API documentation. There are many different values that we don't need for our system, only the content of the message is what we need. So, we construct the data model to represent each key that we need in three Swift structs which are nested, shown in Figure 4-7. Because this time all the names of the property are exactly the same as the key is in the JSON, so we don't have to define the custom CodingKeys and initializers, and the Codable protocol will take care of everything.

4.2.3 System and User Prompt Composing

Currently there are two main types of models provided by OpenAI that has two different tiers of performance. GPT-3 and its series are the first set of models that are open to the public, which is slightly older. GPT-3.5 is the model with similar performance regarding content quality but with a significantly faster speed. GPT-4 series is the new-comer which have much better performance quality wise, but some earlier models are much slower. GPT-4-turbo and GPT-4o are the newest models released by OpenAI this year that have improved performance speed wise. In terms of price, the GPT-3 series cost much less than GPT-4 series with just 0.5 dollars per 1 million tokens, while the GPT-4 costs 10 times the price. Because iJourney requires a very strict content, where only JSON can be accepted as the output, the content quality is a bigger concern than speed or price, considering that Microsoft is generous enough to give 100 dollars to us as students to use the API service from OpenAI. So the GPT-4 series will be our primary choice of models.

Before we can use ChatGPT as our JSON generator, we need to compose a robust set of system and user prompt to ensure the quality of the response results. OpenAI

kindly provided a tool called GPT builder to create your own GPT in the GPTs section of their service. On there, we can use another GPT to help us create the prompt set. When building the GPT in the GPTs, the model would default to be the newest GPT-4 model. As the writing time of this thesis, the newest GPT model is GPT-4-Turbo-04-09, which will be our choice for the system.

As shown in Figure 4-8, this is the prepared system prompt for GPT-4 turbo. This prompt emphasis the importance of a structured response from the model which is crucial for our system since during the decoding process, we cannot process natural languages directly and a JSON structure is required. This is somewhat like a role-play-based prompt where we assign to the GPT as a trip itinerary creator assistant.

```
As a Trip Itinerary Creator, I specialize in designing custom travel plans
tailored to individual preferences and needs, now presenting itineraries in a
refined JSON output. When provided with specific details such as destination,
itinerary length and interests, I compile a detailed plan. The updated structure
showcases itineraries as a list, where each day includes up to three points of
interest (POIs). Each POI is detailed with its name, type, description.

I ensure the itinerary matches the user's interests, sticking strictly to this
JSON format. In situations where the input lacks detail, I'll return "Internal
error" instead of seeking clarification. My aim is to deliver accurate, practical,
and engaging travel plans through this structured and clear format, always with a
professional demeanor reflective of my travel planning expertise. I prioritize
creating a balanced itinerary over covering all personal interest types, focusing
on quality and diversity within the limit of up to three POIs per day.

OUTPUT template in JSON:
{city_name: <City Name>,country_name: <Country Name>,itinerary_list:[{poi_list:
[{name: <POI Name>,type: <POI Type>,description: <POI Description>},...]},...]}
```

Figure 4-8 System Prompt

As shown in Figure 4-9, this is the user prompt in the iJourney system. It is called the user prompt, but in our system, this is actually not sent by the user, rather, it will be sent by the system in this fixed format. The information that this prompt contains will be gathered during the itinerary creation process when the user will input all the necessary information. Then the system will put that information into this format to generate a complete user prompt. Then the user prompt can be sent to ChatGPT. During the process, the user will have no idea about the actual prompt. By doing this, we can ensure that the ChatGPT will never receive a faulty user prompt that may potentially lead to a faulty response. This is the main difference between our system and other chat-based system, where we basically eliminate the exposure to prompt injection and other prompt security risks.

```
Trip Plan: {
    Target Country: <CountryName>,
    Target City: <CityName>,
    Itinerary Length: <n> Days,
    Personal Interests: {
        Historical: <InterestLevel>,
        Cultural: <InterestLevel>,
        Nature: <InterestLevel>,
        Entertainment: <InterestLevel>,
        Shopping: <InterestLevel>,
        Sports: <InterestLevel>,
        Adventure: <InterestLevel>,
        Relaxation: <InterestLevel>,
        Food: <InterestLevel>,
        Nightlife: <InterestLevel>,
    }
}
```

Figure 4-9 User Prompt Template

4.2.4 Mockup Data Source

Before we implement the actual features into the system, we need a mockup data secure so that we can test everything locally. This will save our time and effort during the development process and if anything goes wrong, we will be able to fix or change them quickly without messing with too much network which itself will bring much more variables. Also, in order for us to implement the feature of searching a city in a country, we will have to have the full list of city and country in the world.

For the full list of city and country in the world, considering that the current world is relatively stable that the name and location of a city or a country would basically stay the same, it would be a simple solution that we just use local data instead of relying on third-party real-time databases. So we found a GitHub repo that has few different formats of JSON files that has most of the cities and countries in the world (80000+ cities), and we pull the JSON files and hard-coded into our system. When the system needs the data, it will just load the JSON files locally. This saves time and network resources.

As for the itinerary data, we created two sets of mockup data. One is a JSON of an itinerary, which we used to test if the JSON parser works in the system, as well as test other itinerary related information display. And we also created a mockup ChatGPT response data in JSON format so that we can test network related functions in the system.

4.2.5 Networking and API

Our system is a client only structure, which means every third-party network requests have to be done on-device rather than rely on a relay server. Thus, we need to deal with all sorts of network requests as well as its testing before getting it online.

To make the network module testable, we created a protocol called NetworkService, see Figure 4-10. It has one required protocol function called fetchData() where it takes a URLRequest object and return a Data object, but it's not implemented because we will utilize the inheritance and polymorphism of Swift. Then we extend the URLSession class to inherit the NetworkService protocol and implement the fetchData() method to create a real HTTP request. When we want to test the function where it has implement the fetchData() method, we can create a test class which inherits the same NetworkProtocol to create a fake HTTP request.

```swift
protocol NetworkService {
  func fetchData(with request: URLRequest) async throws -> Data
}

extension URLSession: NetworkService {
  func fetchData(with request: URLRequest) async throws -> Data {
    let (data, response) = try await self.data(for: request)
    guard let httpResponse = response as? HTTPURLResponse,
  (200...299).contains(httpResponse.statusCode) else {
      throw ItineraryError.networkError
    }
    return data
  }
}
```

Figure 4-10 NetworkService Protocol

The HTTP request itself is rather simple since we will only use the standard HTTP protocol to communicate between the system and the APIs.

In the network module, when we need to implement each service class for each network request operations, we need to bring one network service object with it as a class property. For example, see Figure 4-11, this is the initial part of the ItineraryService class, which will be responsible for communicate with ChatGPT API. We have the networkService as a private member, and initialize the networkService in the init() function where we assign the default inheritance is URLSession, which will send the real request when in use. When we want to test the service class to see if there's any problems, we can simply assign a test network service when we create the object with the initializer. All other service classes in the system are implemented likewise.

```swift
class ItineraryService {

  private let networkService: NetworkService

  private let endpoint: String
  private let apiKey: String
  private let systemPrompt: String

  init(networkService: NetworkService = URLSession.shared) {
    self.networkService = networkService
    guard let endpoint = APIConfiguration.shared.chatGPTEndpoint,
          let apiKey = APIConfiguration.shared.chatGPTAPIKey,
          let systemPrompt = APIConfiguration.shared.systemPrompt else {
      fatalError("API not found")
    }
    self.endpoint = endpoint
    self.apiKey = apiKey
    self.systemPrompt = systemPrompt
  }

}
```

Figure 4-11 ItineraryService Class

Because we are using API keys to access all the resources, including ChatGPT's API, we will need a way to store the API keys so that it's not hard-coded and can be easily replace during the development process. We created a plist file in our project where the API keys are several key-value pairs, and we load the plist in a dedicated APIConfiguration class and make it accessible to other classes. Then whenever we need the keys, such as in the init() function in Figure 4-11, we can fetch the key from the APIConfiguration class.

During the network testing process, we also need a way to see if we are sending or receiving the correct messages or not before we do anything in the code. So, we used Proxyman as our network diagnostic tool. After installing the certificate on the system and also on the iOS simulator, we can decrypt the HTTPS messages. Using the tool, we can ensure that the system send the correct format of information to the ChatGPT API and also ensure that ChatGPT returns the desired results with the correct format. See Figure 4-12, this is one of the captured HTTPS requests and responses during a testing environment.

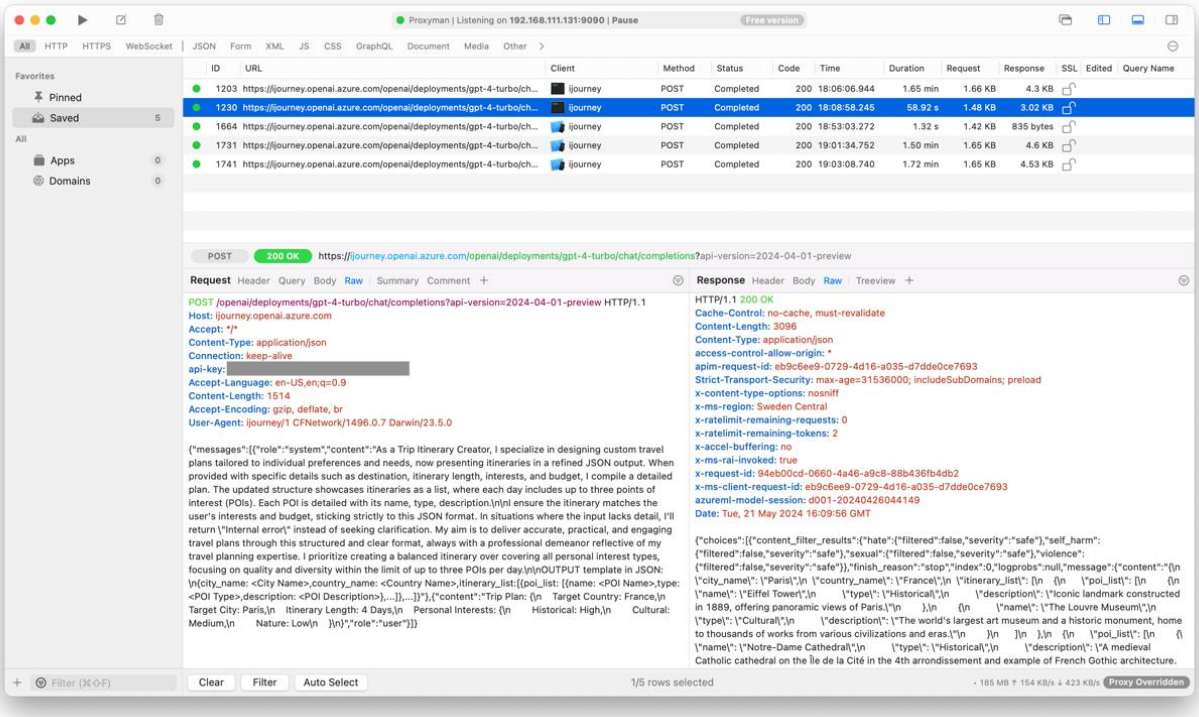

Figure 4-12 Proxyman Network Tool

4.2.6 MapKit Integration

As shown in Figure 4-13, this is the function where the system will be triggered when the user tapped “View in Maps” button. The function will take the system and jump to Maps app and navigate to where the POI is.

The openInMaps function searches for a point of interest (POI) on a map using natural language queries composed of the POI's name, the city name, and the country name from an itinerary. It initializes an MKLocalSearch request with these parameters and sets the search region to cover the entire world. The search is executed asynchronously, and the response is processed to filter map items that match the exact city and country names. If a matching item is found, it opens the location in the Maps app, centered on the POI's coordinates with a specific zoom level. If there are no matching results or an error occurs, appropriate messages are printed.

```swift
private func openInMaps() {
    let request = MKLocalSearch.Request()
    request.naturalLanguageQuery = "\(poi.name), \(itinerary.cityName!), \
(itinerary.countryName!)"
    request.region = MKCoordinateRegion(MKMapRect.world)

    let search = MKLocalSearch(request: request)
    search.start { response, error in
      guard let response = response else {
        print("MKLocalSearch response error")
        return
      }

      // Filter results to find the POI with the exact name, city, and country
      let filteredItems = response.mapItems.filter { item in
        if let name = item.placemark.name,
           let locality = item.placemark.locality,
           let country = item.placemark.country {
          return locality == itinerary.cityName && country ==
itinerary.countryName
        }
        return false
      }

      guard let mapItem = filteredItems.first else {
        print("No matching results for the POI")
        return
      }

      mapItem.openInMaps(launchOptions: [
        MKLaunchOptionsMapCenterKey: NSValue(mkCoordinate:
mapItem.placemark.coordinate),
        MKLaunchOptionsMapSpanKey: NSValue(mkCoordinateSpan:
MKCoordinateSpan(latitudeDelta: 0.0035, longitudeDelta: 0.0035))
      ])
    }
  }
```

Figure 4-13 Open in Maps Function

4.3 Unit and Integration Test

We created several test classes to test different functions of the system. We can take the itinerary related test as an example. As shown in Figure 4-14, this is one of the unit tests of itinerary service. This can test whether the ItineraryService class works well or not. We used a test network service to make fake network requests so that we can separate network related issue with the system as we are testing it. The test network service will return a fixed response locally where we could assert if the itinerary fetched by itinerary service is correct or not.

We will also need a way to test if the real network connection works or not, and this needs to be done by an integration test because we will test all the functions in the itinerary service all at once, see in Figure 4-15. This test will send a real network request, and test everything in the itinerary service but without the user interface.

```swift
import XCTest
@testable import ijourney

final class ItineraryServiceTests: XCTestCase {

  func testServiceDoesFetchNewItineraryData() async throws {
    let networkService = TestItineraryNetworkService()
    let service = ItineraryService(networkService: networkService)
    let itinerary = try await service.fetchItinerary(userPrompt: "")

    XCTAssertEqual(itinerary.days.count, 4)
  }
}
```

Figure 4-14 Itinerary Unit Test

```swift
import XCTest
@testable import ijourney

final class ItineraryServiceIntegrationTests: XCTestCase {

  var service: ItineraryService!

  override func setUp() {
    super.setUp()
    service = ItineraryService()
  }

  func testFetchItinerarySuccess() async throws {

    let sampleCity = City(name: "Paris",
                          countryCode: "FR",
                          countryName: "France")

    var sampleInterests = Interests()
    sampleInterests[.historical] = .high
    sampleInterests[.cultural] = .medium
    sampleInterests[.nature] = .low

    let userPrompt = service.composeUserPrompt(city: sampleCity, startDate:
Date(), endDate: Calendar.current.date(byAdding: .day, value: 3, to: Date())!,
interests: sampleInterests)

    do {
      let itinerary = try await service.fetchItinerary(userPrompt: userPrompt)

      print(itinerary)

      XCTAssertEqual(itinerary.cityName, "Paris")
      XCTAssertEqual(itinerary.countryName, "France")
      XCTAssertEqual(itinerary.days.count, 4)
    } catch {
      XCTFail("Failed to fetch itinerary: \(error)")
    }
  }
}
```

Figure 4-15 Itinerary Integration Test

4.4 Chapter Summary

This chapter focuses on the implementation phase of the AI-assisted travel planning app, iJourney. It begins with the setup of the development environment, detailing the hardware, software, and tools used. The chapter then explores key technologies and implementation strategies, including the structure of data models, JSON parsing, and mapping in Swift, and the composition of system and user prompts. Additionally, it discusses the creation of mockup data sources to facilitate local testing and the handling of networking and API interactions. Finally, the chapter covers unit and integration testing to ensure the system functions correctly and reliably.

CHAPTER 5 CONCLUSION

5.1 Conclusion

The development of the AI-Assisted Travel Planning App, iJourney, demonstrates the potential of integrating advanced AI technologies with modern mobile application frameworks to enhance user experiences in travel planning. This thesis explored the design and implementation of iJourney, an iOS application that utilizes OpenAI's ChatGPT to generate personalized travel itineraries based on user preferences and interests.

The project began with a comprehensive review of current research and technologies related to AI and travel planning. This identified the capabilities of ChatGPT in natural language processing and its applications in various industries, including the travel sector, which is where we are. The study also highlighted the advantages of using the MVVM architecture pattern with SwiftUI in iOS development, emphasizing the benefits of modularity, maintainability, and testability.

System design was meticulously planned, starting with requirement analysis that outlined domain and product-level needs. The requirement analysis pointed out key needs of the users and laid down the key functionalities. Based on what we learned about the MVVM architecture, we designed the architecture of the system using building blocks views. The prototype phase provided a visual representation of the app's interface and navigation hierarchy, adhering to Apple's Human Interface Guidelines. The prototype phase also presented many key functionalities with real interfaces.

Implementation involved setting up a modern development environment and employing key technologies such as Swift, SwiftUI, and MapKit. The development process also incorporated advanced techniques for JSON parsing, networking, and API integration to ensure efficient communication with the ChatGPT API. Unit and integration tests were conducted to verify the functionality and reliability of the system.

The iJourney app successfully showcases how AI can be used to offer dynamic and personalized travel planning solutions. By allowing users to input their travel preferences and interests, the app generates detailed itineraries that have detailed information for each POI, as well as the convenience to see on the map and get directions with just a tap.

In conclusion, the development of iJourney highlights the transformative potential

of combining AI with mobile technology. With the combination of the two, we can really take advantages of the power of AI and the convenience of mobile phones.

5.2 Feature Work

While the development of iJourney has successfully demonstrated the potential of AI in enhancing travel planning, there are several areas for future work that can further improve the application and expand its capabilities.

1. Real City Page Recommendation Algorithm: Currently, iJourney offers a list of featured cities, but implementing a recommendation algorithm based on user behavior and preferences can enhance personalization. By analyzing user interactions and preferences, the app can provide more relevant city recommendations, making the exploration process more intuitive and engaging.

2. Local Data Set to Train the LLM: To improve the accuracy and relevance of itinerary suggestions, a local dataset could be created to train the language model. This dataset would include specific information about local attractions, events, and user reviews. Training the model with this data can result in more precise and context-aware recommendations.

3. More Information about Each Place Using Travel Affiliated Data Sources: Integrating additional travel-related data sources, such as partnerships with travel agencies, tourism boards, and review platforms, can enrich the information provided about each point of interest. This can include detailed descriptions, user reviews, ratings, and insider tips, offering users a comprehensive view of their destinations.

4. SSO Login Implementation: Implementing Single Sign-On (SSO) login options, such as Google or Apple accounts, can streamline the user authentication process. This feature can enhance user convenience, improve security, and facilitate easier access to personalized content across different devices.

5. Enhanced User Interface and Experience: Continual refinement of the user interface and experience, based on user feedback and usability testing, can further enhance the app's appeal and functionality. Future updates could include more interactive maps, richer visual content, and improved navigation flows to make the app even more user-friendly.

6. Expansion to Other Platforms: While iJourney is currently designed for iOS, expanding its availability to other platforms such as Android and web browsers can broaden its user base. Cross-platform development frameworks or dedicated native apps for each platform can be considered to ensure a seamless user experience across

devices.

7. Incorporating User-Generated Content: Allowing users to share their itineraries, reviews, and travel experiences within the app can create a community-driven platform. This user-generated content can provide valuable insights and inspiration for other travelers, enhancing the overall utility and engagement of the app.

8. Advanced AI Features: Exploring advanced AI features such as voice interaction, real-time itinerary adjustments, and augmented reality (AR) for on-site navigation can take the app to the next level. These features can offer a more immersive and responsive travel planning experience, catering to the evolving needs of modern travelers.

9. Inter-Itinerary and Inter-POI Transport Recommendation: Currently iJourney can only generate an itinerary with a list of point of interests without any transportation information. The user can search the transportation by tapping “View In Maps” when viewing the POI, but this takes extra effort. Adding transportation recommendations between different itineraries and points of interest can significantly enhance the user's travel experience. This feature can provide users with options for the best modes of transport, estimated travel times, and cost-effective routes, making their travel plans more efficient and seamless.

These future enhancements will build on the solid foundation established by iJourney, driving its evolution into a more comprehensive and intelligent travel planning tool. By continuously integrating user feedback and leveraging the latest technological advancements, iJourney can remain at the forefront of innovation in the travel industry.

Reference

[1] H. Naveed *et al.*, "A Comprehensive Overview of Large Language Models," 2023, doi: 10.48550/ARXIV.2307.06435.

[2] J. Kaddour, J. Harris, M. Mozes, H. Bradley, R. Raileanu, and R. McHardy, "Challenges and Applications of Large Language Models," 2023, doi: 10.48550/ARXIV.2307.10169.

[3] B. Chen, Z. Zhang, N. Langrené, and S. Zhu, "Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review." arXiv, Oct. 27, 2023. Accessed: Mar. 29, 2024. [Online]. Available: http://arxiv.org/abs/2310.14735

[4] T. Wu *et al.*, "A Brief Overview of ChatGPT: The History, Status Quo and Potential Future Development," *IEEE/CAA J. Autom. Sinica*, vol. 10, no. 5, pp. 1122–1136, May 2023, doi: 10.1109/JAS.2023.123618.

[5] A. Nazir and Z. Wang, "A comprehensive survey of ChatGPT: Advancements, applications, prospects, and challenges," *Meta-Radiology*, vol. 1, no. 2, p. 100022, Sep. 2023, doi: 10.1016/j.metrad.2023.100022.

[6] C. M. Gallardo Paredes, C. Machuca, and Y. M. Semblantes Claudio, "ChatGPT API: Brief overview and integration in Software Development," *Int. Jour. Eng. Ins.*, vol. 1, no. 1, pp. 25–29, Nov. 2023, doi: 10.61961/injei.v1i1.7.

[7] A. Bahrini *et al.*, "ChatGPT: Applications, Opportunities, and Threats," in *2023 Systems and Information Engineering Design Symposium (SIEDS)*, Charlottesville, VA, USA: IEEE, Apr. 2023, pp. 274–279. doi: 10.1109/SIEDS58326.2023.10137850.

[8] M. N.-U.-R. Chowdhury and A. Haque, "ChatGPT: Its Applications and Limitations," in *2023 3rd International Conference on Intelligent Technologies (CONIT)*, Hubli, India: IEEE, Jun. 2023, pp. 1–7. doi: 10.1109/CONIT59222.2023.10205621.

[9] F. Fui-Hoon Nah, R. Zheng, J. Cai, K. Siau, and L. Chen, "Generative AI and ChatGPT: Applications, challenges, and AI-human collaboration," *Journal of Information Technology Case and Application Research*, vol. 25, no. 3, pp. 277–304, Jul. 2023, doi: 10.1080/15228053.2023.2233814.

[10] R. L. Logan IV, I. Balažević, E. Wallace, F. Petroni, S. Singh, and S. Riedel, "Cutting Down on Prompts and Parameters: Simple Few-Shot Learning with Language Models." arXiv, Jul. 01, 2021. Accessed: Apr. 08, 2024. [Online]. Available: http://arxiv.org/abs/2106.13353

[11] V. Sanh *et al.*, "Multitask Prompted Training Enables Zero-Shot Task Generalization." arXiv, Mar. 17, 2022. Accessed: Apr. 08, 2024. [Online]. Available: http://arxiv.org/abs/2110.08207

[12] A. Webson and E. Pavlick, "Do Prompt-Based Models Really Understand the Meaning of their Prompts?" arXiv, Apr. 21, 2022. Accessed: Apr. 08, 2024. [Online]. Available: http://arxiv.org/abs/2109.01247

[13] S. Arora *et al.*, “Ask Me Anything: A simple strategy for prompting language models,” in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=bhUPJnS2g0X

[14] Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp, “Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity.” arXiv, Mar. 03, 2022. Accessed: Apr. 08, 2024. [Online]. Available: http://arxiv.org/abs/2104.08786

[15] D. Gursoy, Y. Li, and H. Song, “ChatGPT and the hospitality and tourism industry: an overview of current trends and future research directions,” *Journal of Hospitality Marketing & Management*, vol. 32, no. 5, pp. 579–592, Jul. 2023, doi: 10.1080/19368623.2023.2211993.

[16] F. Sudirjo, P. Diawati, Y. Riady, A. M. Almaududi Ausat, and S. Suherlan, “The Role of ChatGPT in Enhancing the Information Search and Decision-Making Process of Travellers,” *jmp*, vol. 12, no. 1, pp. 500–507, May 2023, doi: 10.33395/jmp.v12i1.12443.

[17] M. Demir and Ş. Ş. Demir, “Is ChatGPT the right technology for service individualization and value co-creation? evidence from the travel industry,” *Journal of Travel & Tourism Marketing*, vol. 40, no. 5, pp. 383–398, Jun. 2023, doi: 10.1080/10548408.2023.2255884.

[18] M. Kovács and Z. C. Johanyák, “Comparative Analysis of Native and Cross-Platform iOS Application Development,” *Műszaki Tudományos Közlemények*, vol. 15, no. 1, pp. 61–64, Oct. 2021, doi: 10.33894/mtk-2021.15.12.

[19] R. B. Paramadani, M. A. Akbar, and A. Pinandito, “User Interface Rendering Time in Android Applications: Revealing the Effects of Design Patterns,” in *Proceedings of the 8th International Conference on Sustainable Information Engineering and Technology*, Badung, Bali Indonesia: ACM, Oct. 2023, pp. 631–636. doi: 10.1145/3626641.3626948.

[20] A. Wilson, F. Wedyan, and S. Omari, “An Empirical Evaluation and Comparison of the Impact of MVVM and MVC GUI Driven Application Architectures on Maintainability and Testability,” in *2022 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*, San Antonio, TX, USA: IEEE, Sep. 2022, pp. 101–108. doi: 10.1109/IDSTA55301.2022.9923083.

[21] H. Magics-Verkman, D. R. Zmaranda, C. A. Győrödi, and R.-Ş. Győrödi, “A Comparison of Architectural Patterns for Testability and Performance Quality for iOS Mobile Applications Development,” in *2023 17th International Conference on Engineering of Modern Electric Systems (EMES)*, Oradea, Romania: IEEE, Jun. 2023, pp. 1–4. doi: 10.1109/EMES58375.2023.10171619.

[22] T. Brown *et al.*, “Language Models are Few-Shot Learners,” in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[23] A. Inc, “SwiftUI Overview - Xcode - Apple Developer.” Accessed: Apr. 17, 2024. [Online]. Available: https://developer.apple.com/xcode/swiftui/
[24] “MapKit,” Apple Developer Documentation. Accessed: Apr. 10, 2024. [Online]. Available: https://developer.apple.com/documentation/mapkit/
[25] “Introducing GPTs.” Accessed: Apr. 11, 2024. [Online]. Available: https://openai.com/blog/introducing-gpts

Acknowledgment

Now this is nearing the end of my undergraduate thesis work. Admittedly, it took me a long time to complete my thesis, but I have gained a lot from this work. I have learned a lot in the field of iOS development and the new rising language models.

I want to thank all my teachers, counselors, and everyone guided me along the way both in Zhejiang University of Technology and Blekinge Institute of Technology. I want to thank Professor Li Xiaoxin as being my thesis supervisor, providing valuable advice and help during my study. I also want to thank Yu Shulan as being my counselor, helping me with various matters in many situations.

Lastly, I want to thank my family who gave me the strongest support and my friends who have been with me along the way. Without them, I could never go this far in my study and life journey.

Appendix

附件 1 毕业设计开题报告

附件 2 毕业设计文献综述

附件 3 毕业设计外文翻译（中文译文与外文原文）