Software Requirements Specification and Design document

for

<Tourist Info>

Version 1.0 approved

Prepared by Zach Miller, Aaron Cambridge, Rebecca Hall, River Hallie

CSUSM CS 370

November 7, 2024

Table of Contents

Table of Contents ii

Revision History iii

1. Introduction 1

1.1 Purpose 1

1.2 Document Conventions 1

1.3 About our project team 1

1.4 Intended Audience and Reading Suggestions 1

1.5 Product Scope 1

1.6 References 1

2. Statement of Work 2

2.1 Communication 2

2.2 Dependencies and Constraints 2

2.3 Design, Development, and Implementation Methods 2

2.4 Change Management 2

3. Timeline 2

4. Overall Description 3

4.1 Product Perspective 3

4.2 Product Functions 3

4.3 Technical requirements 3

4.4 Operating Environment 3

4.5 Design and Implementation Constraints 3

4.6 User Documentation 3

4.7 Assumptions and Dependencies 4

5. External Interface Requirements 4

5.1 User Interfaces 4

5.2 Hardware Interfaces 4

5.3 Software Interfaces 4

5.4 Communications Interfaces 4

6. System Features / Functonal Requirements 5

6.1 System Feature 1 5

6.2 System Feature 2 (and so on) 5

7. Nonfunctional Requirements 5

7.1 Performance Requirements 5

7.2 Safety Requirements 6

7.3 Security Requirements 6

7.4 Software Quality Attributes 6

7.5 Business Rules 6

8. System Architecture 6

8.1 Architectural Design 6

8.2 Decomposition Design 6

8.3 Design Rationale 7

9. Data Design 7

9.1 Entity relationship design 7

9.2 Data dictionary 7

10. Other Requirements 7

Appendix A: Screen Images 7

Appendix B: Analysis Models 7

Appendix C: To Be Determined List 7

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Aaron Cambridge,  Rebecca Hall,  River Hallie,  Zach Miller | 9/19/2024 | initial draft | 1.0 draft 1 |
| Aaron Cambridge,  Rebecca Hall,  River Hallie,  Zach Miller | 11/7/2024 | Milestone draft | 1.2 draft |

# Introduction

## Purpose

This Software Requirement Specification (SRS) document outlines the software requirements for Tourist Info, specifically version 1.2. The project integrates an API that enables users to input a location and receive details about local attractions and places of interest. The chosen API offers extensive features, including place details, nearby searches, and recommendations based on user preferences. This SRS emphasizes the functionalities of the API, how user decisions will interact with these features, and the structured presentation of results. The document also covers both functional and non-functional requirements to support the Tourist Info application's goals throughout the software development lifecycle.

## Document Conventions

**Bold text** is used to highlight key terms, section headings and important elements.

## About our project team

ZARR is a group of four computer science majors taking an Introduction to Software Engineering Course (CS 370) at California State University San Marcos (CSUSM). Our team is dedicated to making quality software for our clients.

## Intended Audience and Reading Suggestions

This document is intended for potential clients, developers, users, and other stakeholders interested in using or understanding what our product does and how it functions. It covers the product’s scope, dependencies and constraints, design methods, and an overall description of the project. It is suggested that readers read this document in the order in which it is listed.

## Product Scope

Our Tourist Info project is intended to provide users with a convenient place to explore top tourist destinations throughout the world to visit and explore. They will be able to input a city or country they are interested in visiting as well as the type of attraction they would like to see and get recommendations to consider. If the user is unsure, we would also have a list of the top tourist destinations in the world to give ideas of places they may want to explore further through our software.

## References

API used: <https://rapidapi.com/gmapplatform/api/google-map-places>

GitHub repository: https://github.com/AaronCSUSM/ZARR\_GroupProject

# Statement of Work

## Communication

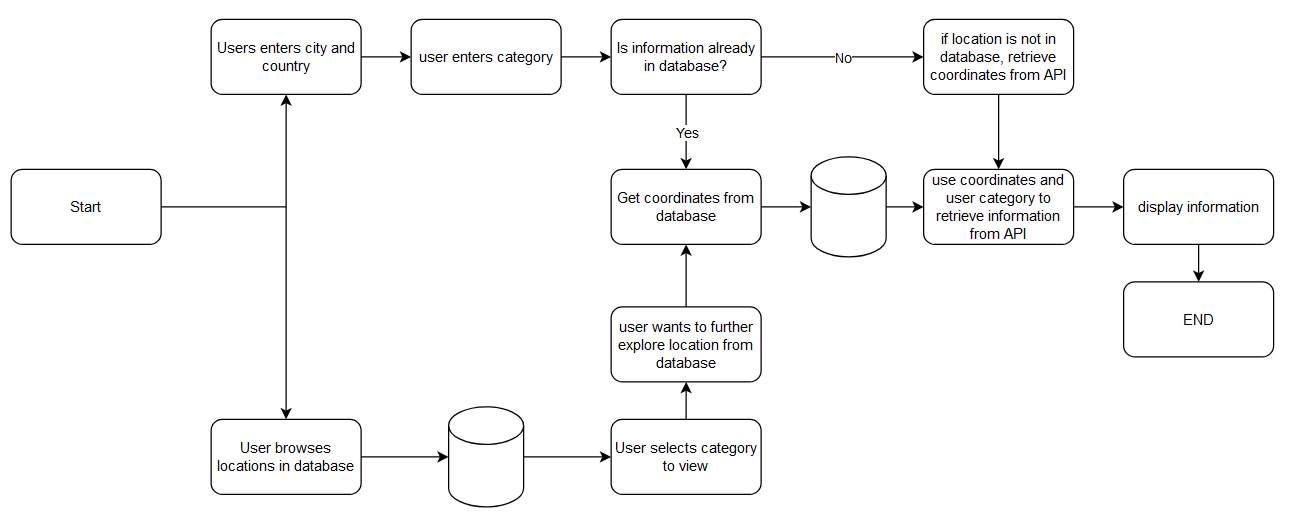
This group has determined that to maintain effective communication throughout our tourist info project, we will manage communication by combining in-person meetings as well as online meetings to talk over issues and updates that are urgent. There will be weekly check-ins that will happen two times every week, where we will go over our progress, discussing challenges that we are facing to see how we can get over it, and planning for the next steps to take. We will continue to be adaptable as a team when it comes to setting up meetings during the day to discuss urgent issues that arise and how we can fix them. Discord will be our main form of communication for team discussions, updates, and urgent issues quickly. We will stay connected throughout the future weeks to make sure everyone is aligned with the project goals.

## Dependencies and Constraints

This project will be coded in Java using Eclipse as the IDE. The graphical user interface for the product will be created with Java Swing and the database will be managed with Java DataBase Connectivity (JDBC). We will be using GitHub to handle version control. This product will also be relying on Google Map Places as an external API to provide accurate data in real time.

## Design, Development, and Implementation Methods

Our development group will have an agile methodology, to utilize iterative and incremental development for building the product. A timetable will be used to manage and allocate tasks effectively throughout the project timeline. This project will be developed in a Java environment, while focusing on ensuring smooth integration between the database and the server. Our plans are to use a Java IDE as well as testing on individual functions, objects, and at the connection points between the server and database. For source code control, we will employ Git to manage version control and collaborative development effectively. Quality assurance will involve code reviews and automated testing to maintain high standards of functionality and stability. Backup and recovery procedures will be established for the source code and design documents to prevent data loss. All of the design documentation will be owned by the development team with regular reviews from the client to make sure the requirements are being met along with getting feedback.



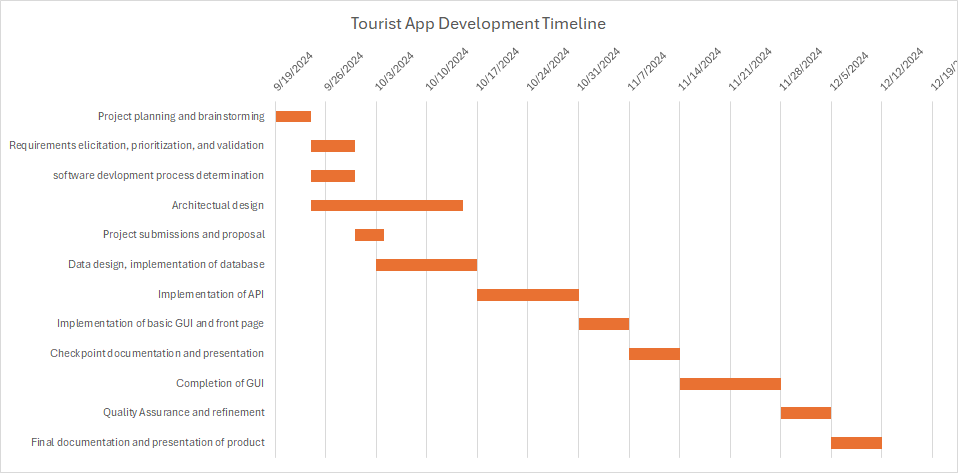
## Change Management

Changes to the project scope, requirements, technology, or contract will be proposed and reviewed by both the development team and the client. Each change request will outline its purpose and impact on schedule, cost, or quality. If changes are approved then it will result in updates to the project time and budget, with adjustments to communications to the stakeholders.

Currently no changes to the product are expected. It is anticipated that any changes that are made will be minor in scope and not impact on the cost or schedule estimates. In the unlikely event that there are major changes that need to be made at this stage in the product’s development, they will be discussed and approved by the client first.

# Timeline

|  |  |  |
| --- | --- | --- |
| Phase | Description | Estimated completion date. |
| 1 | Project proposal and presentation | 9/24/2024 |
| 2 | Requirements elicitation, prioritization, and validation | 9/30/2024 |
| 3 | Software development process determination | 9/30/2024 |
| 4 | Architectural design | 9/14/2024 |
| 5 | Project submissions and proposal | 10/3/2024 |
| 6 | Data design, implementation of database | 10/17/2024 |
| 7 | Implementation of API | 10/31/2024 |
| 8 | Implementation of basic GUI and front page | 11/7/2024 |
| 9 | Checkpoint documentation and presentation | 11/14/2024 |
| 10 | Completion of GUI | 11/28/2024 |
| 11 | Quality Assurance and refinement | 12/5/2024 |
| 12 | Final Documentation and presentation of product | 12/12/2024 |



# Overall Description

## Product Perspective

Our Tourist Info project being specified in this SRS is a new self-made product designed to provide users with relevant local attractions and place-based information based on their input. The product uses an API to retrieve data about places and attractions in a specified location. This system doesn’t replace any existing system, but it does aim to fill a gap in providing real-time and customizable location-based information for tourists. It is a standalone application, but the system does rely on the API for retrieving place data and details. The API is a crucial part of the project’s functionality by ensuring accurate and up-to-date information is available to the users. If this product is integrated with a larger system, it could serve as a module providing location-specific information. The interface between this system and any other platform would rely on the seamless integration of API calls.

## Product Functions

The primary function of this product is to allow users to submit a city and country they would like to visit or vacation to, as well as a category they are interested in (e.g. museums, beaches, parks) and receive back recommendations and some information of places that fit their criteria. The secondary function of this product is to provide the user with recommendations of popular tourist destinations and attractions based on data from recent years, at which point they can then use the products primary function to further investigate any place that piques their interest.  
  
Major Functions of Tourist Info:

* Primary Function: Allow users to enter a city and country along with a category (e.g., museums, beaches, parks) to receive customized recommendations for attractions matching their criteria.

* Secondary Function: Provide users with general recommendations for popular tourist destinations and attractions based on recent data, allowing further exploration through the primary function.

## Technical requirements

Our product would require the user to own a computer with an operating system that supports modern web browsers. The device the user owns must have sufficient storage space for offline functionality. The system will also need a stable internet connection to be able to access the APIs used in this product. The development environment for the product needs a system capable of running Java, and access to IDEs such as Eclipse.

System and Software Requirements

Mandatory Requirements:

* Users must have a computer running an operating system that supports modern web browsers.
* The user's device must have sufficient storage space to support offline functionality.
* A stable internet connection is required to access the APIs used by the product.
* The system must be capable of running Java to support development and operation.
* The development environment must include access to Java IDEs such as Eclipse.

## Operating Environment

This software will operate on desktops or laptops that are compatible with modern operating system such as Windows 10 or Windows 11. It will also require a web browser and stable internet connection to connect to external APIs.

## Design and Implementation Constraints

The primary constraint of this product is the timing, as the deadline will limit the number of features we could otherwise add. Nevertheless, our team is confident in producing a stable product with the required features by the specified deadline. The use of Java presents a minor design constraint due to our team’s varying levels of experience with this programming language, however we believe adequate communication and practice will prevent this constraint from impacting the final product. Finally, the requirement of an external API will impact the type of data we can incorporate and utilize in our product, depending on the API’s capabilities and limitations. No other constraints are known at this time.

## User Documentation

This SRS form is currently the only documentation available. A user manual for the final product can be provided by the completion date if the client requests one.

## Assumptions and Dependencies

Our assumptions for this project are that the external API we intend to use, Google Map Places, will be functioning correctly and be available during the duration of the project. We are making the same assumption for GitHub to manage version control for our team. It is assumed that users will have a stable internet connection, as the program relies on online data retrieval from external sources. We also assume a stable Java development environment compatible with IDEs like Eclipse, as any major changes in tools or dependencies could disrupt development and testing.

# External Interface Requirements

## User Interfaces

The primary interface for the Tourist Info application is a homepage with a simple, visually organized layout, that will be shown. The interface follows a basic structure to ensure ease of navigation and accessibility. Key elements include:

* Screen Layout: The homepage features a title at the top, “Time to take a trip!,” followed by two main buttons, “Search” and “Random Recommendation,” which allow users to either search for a specific destination or receive a random travel suggestion. Below these buttons are four category buttons for quick access to “5 Best Beaches,” “5 Best Amusement Parks,” “5 Best Landmarks,” and “5 Best Museums.”

* Button Functionality: Each button, when clicked, currently displays a placeholder message (“opening: [that page]”) to indicate that a new page would open. Future iterations will navigate users to corresponding pages with relevant content. Each page will also include a “Go Back” button to return to the homepage.
* GUI Standards and Style: The layout is clean and simple for now but will be enhanced in later versions to improve visual appeal. This includes potential adjustments to colors, font styles, and spacing to create a more engaging user experience.

* Navigation and Usability: The interface aims for straightforward navigation, with easily identifiable buttons. Planned additions include back-navigation and standardized button locations to create a consistent user flow.

Further details of the user interface design, including future style guides, color schemes, and additional elements, will be documented separately in the user interface specification.

## Hardware Interfaces

The Tourist Info application is designed primarily for desktop and laptop computers running a modern operating system, such as Windows 10 or Windows 11. The application requires minimal direct interaction with hardware components, as it operates within a standard Java-based environment and relies on a web browser for accessing external APIs. The application retrieves data (such as location and attraction information) from external APIs over the internet. There are no direct hardware data interactions; all data is managed through the software layer, with the application using standard HTTP requests to fetch and display information from API sources. The software communicates with external servers via HTTP/HTTPS protocols to retrieve information from APIs like the Google Maps Places API and Maps Data API. These interactions are essential for the functionality of the application, enabling it to fetch real-time data about tourist destinations.

## Software Interfaces

Our application connects to several key software components and libraries to deliver location-based information to users. These connections are essential for retrieving, processing, and displaying tourist information that users will need. These connections rely on specific APIs and development tools.   
APIs:

* Google Maps Places API (via RapidAPI): This API provides information about tourist attractions, points of interest, and other location-based data. The Tourist Info application sends HTTP requests to this API with parameters like location and category and receives JSON responses containing relevant data for display.
* Development Tools and Libraries: The application is built in a Java environment, using tools like Eclipse IDE for development and Maven for dependency management. Dependencies may include standard Java libraries for handling HTTP requests, and GUI components, ensuring smooth integration with API responses and user interface elements.
* Incoming Data: When a user inputs a location or category (e.g., “beaches”), the application sends a request to the API and receives back JSON-formatted data containing information such as the names, descriptions, and coordinates of relevant attractions.
* Outgoing Data: The application sends requests formatted with HTTP methods and query parameters (e.g., location and category) to retrieve specific data from the APIs.

This integration with external APIs and development tools ensures that Tourist Info can provide timely, accurate information to users while maintaining efficient communication and data handling

## Communications Interfaces

This application relies on network communications to retrieve real-time location-based information from external APIs. These communications are essential for allowing users to receive up-to-date recommendations for tourist attractions based on their input. The current free version of the API allows 100 requests within a 24-hour period.

# System Features

The functional requirements for the Tourist Info application are organized by system features and key use cases to provide a clear and logical understanding of the major services the product will offer. This approach highlights how each feature addresses user needs and defines how the application will operate in various scenarios. Each system feature is broken down by specific use cases to detail the functionality from the user’s perspective.

## Location based search using API

4.1.1 Description and Priority

The API feature will allow users to look up information on specific cities based on the location and criteria they choose. The API will provide the city coordinates if necessary and provide the relevant information.

Benefit: 8

Penalty: 9

Cost: 1 (unless client opts for a paid subscription)

Risk: 3

4.1.2 Stimulus/Response Sequences

User action: User enters a location (city and country) and category and clicks a search button.

Response: The app sends an API request, and the API sends back a list of locations in response for the user to browse.

4.1.3 Functional Requirements

REQ-1: The system must be able to connect to the API.

REQ-2: The app must be able to send requests to the API.

REQ-3: The app must be to account for invalid data from user.

REQ-4: The app must be able to sort the data sent back by the API.

## MySQL Database management

4.1.1 Description and Priority

The database is necessary for storing the coordinate results of prior geocoding searches. It is important to the overall product as it allows it to operate with less API requests as it enables searches to be completed without first needing to retrieve the city coordinates each time a search is initiated.

Benefit: 7

Penalty: 6

Cost: 3 (unless client opts for a paid subscription)

Risk: 2

4.1.2 Stimulus/Response Sequences

User action: User enters a location (city and country) and category and clicks a search button.

Response: The database will be searched to see if the coordinates of the city are already stored and available to use for API calls.

User action: User initiates a search with a city that isn’t already in the database.

Response: The geocoding API requests will return the coordinates of the city, and those coordinates will then be stored for future use.

4.1.3 Functional Requirements

REQ-1: This feature must be able to return the coordinates of stored cities.

REQ-2: This feature must be able to store new city coordinates.

REQ-3: This feature must be able to tell if a city is in the database or not.

## User interface management

4.1.1 Description and Priority

This feature is essential as the product needs to be able to display the results of searches. It also needs to be able to display some default data to give the user a starting point. It will also be necessary for initiating searches.

Benefit: 8

Penalty: 7

Cost: 3 (unless client opts for a paid subscription)

Risk: 6

4.1.2 Stimulus/Response Sequences

User action: User clicks a button for browsing default data.

Response: GUI displays default data in neat pages.

User action: User clicks on button to begin a search.

Response: GUI prompts user to enter a city, country, and category to search for.

User action: User types in search parameters.

Response: GUI collects user input and uses it as arguments for database and API searches.

4.1.3 Functional Requirements

REQ-1: This feature must be able to display search results in a neat and orderly manner.

REQ-2: This feature must have functioning buttons.

REQ-3: This feature must be able to collect user input.

# Other Nonfunctional Requirements

## Performance Requirements

The application must meet specific performance standards to ensure a smooth and responsive user experience. Given that the product relies on external APIs to retrieve location-based data, performance requirements focus on maintaining efficient load times, responsiveness, and data accuracy under various conditions. We want the response time to be within a good time limit. The system should retrieve and display search results fairly well. This is crucial because it makes the user engaged. We also want to have good error handling in case of delays or API errors. The system should display a user-friendly message to indicate that there is a temporary issue.

## Safety Requirements

This application that we are making has minimal safety risks associated with its use, since it's a primarily non-critical information system providing tourism recommendations. However, some safeguards can be implemented to ensure the secure and safe use of our product. Since our product relies on external APIs and user inputs, we will avoid storing sensitive personal data beyond what's necessary for functionality.Other than privacy concerns which we avoid by not storing any user data, there are no strict safety requirements as this software is not intended for critical use such as health care or traffic control.

## Security Requirements

Our application can implement several security and privacy measures to protect user data and make sure there is safe interaction with external services. There can be data privacy which our product will only collect minimal user information necessary to provide accurate recommendations based on location and preferences. We will ask for no sensitive personal data like health information and such. The application as well, won't retain any user input beyond the active session. Another one we can implement would be user Identity authentication. In future versions we can introduce user accounts/profiles with secure authentication methods like having password protection. We can also add general data protection policies which ensure compliance with relevant standards.

## Software Quality Attributes

Usability is a high priority, and in order to satisfy that requirement the user interface will be kept simple and straightforward. Reliability is also a high priority for us, which is why we’ve chosen what we believe is a very reliable API to use as an external party. We are also building this with reusability in mind by storing coordinates made from searches into the database for future use. We are also implementing extensive testing via JUnit to maintain a high quality of testability.

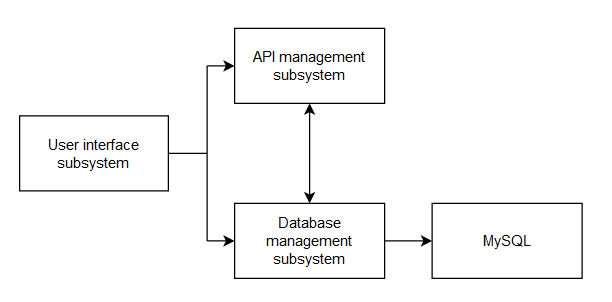
## Business Rules

The owner(s) of the product will be able to add additional city and coordinate data into the database used by the product. Searches made by users for cities that are not already in the database will be automatically added for future use. Users will not be allowed to add cities to the database directly.

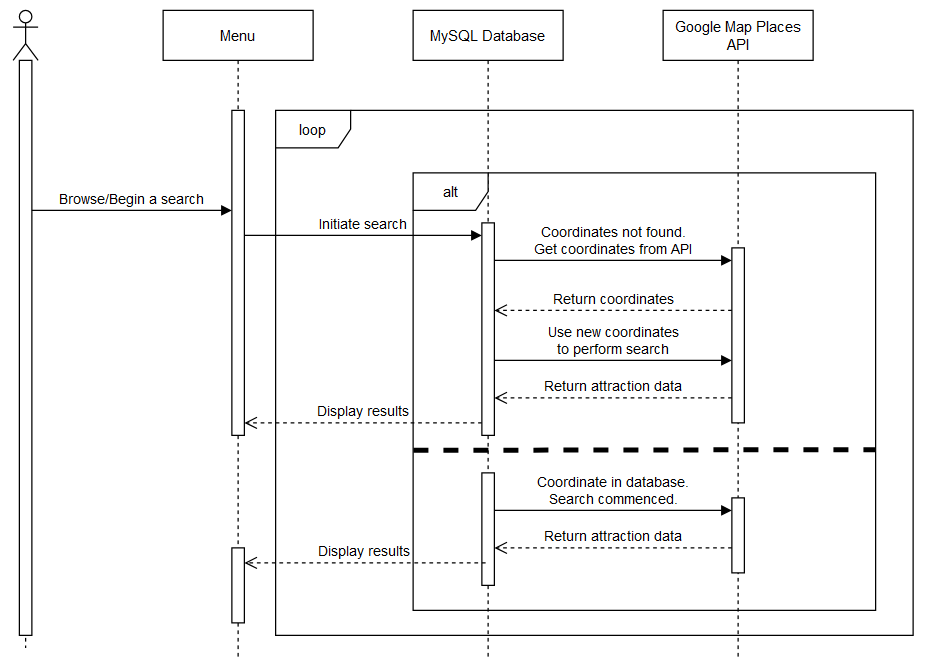
# System Architecture

## Architectural Design

This product is organized into three main subsystems to fulfill its functionality, the user interface subsystem, the API management subsystem, and the database management subsystem. All three of these subsystems interact with each other to allow users to conduct coordinate-based searches while efficiently managing data storage and API calls.

1. The user interface subsystem is the GUI, which will be implemented with Java Swing. This subsystem is responsible for data to the user and allowing them to interact with the database and API management subsystem via search requests.
2. The API subsystem is responsible for finding the data the users request in real time. These include the coordinates of cities and the location details of the area and category the users are interested in.
3. The database management subsystem is responsible for storing the cities and their coordinates. This will be mainly used to reduce the number of API calls needed to perform searches.

## Decomposition Design

Sequence Diagram

## Design Rationale

Our rationale behind choosing the architecture we did was due to the client’s demand that we include them. We did not consider other architecture as we are trying to meet the client’s demands and expectations as closely as possible. Additional features were considered for the product but ultimately rejected due to timing constraints.

# Data Design

## Entity relationship design

## Data dictionary

Database name: zarr\_db

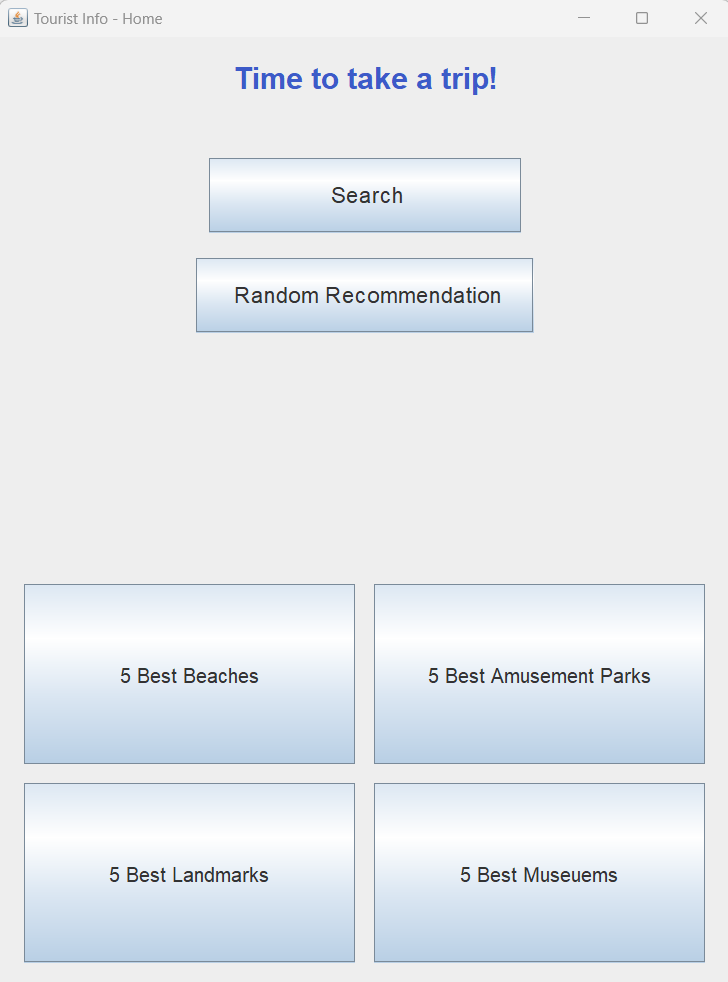
Table name: city\_coordinates

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Id | Int | NO | PRI | NULL | Auto\_increment |
| City | Varchar(100) | NO |  | NULL |  |
| Country | Varchar(100) | NO |  | NULL |  |
| State | Varchar(100) | YES |  | NULL |  |
| Latitude | Decimal(8,2) | NO |  | NULL |  |
| Longitude | Decimal(9,2) | NO |  | NULL |  |

# Other Requirements

This product requires MySQL as a database. There are no legal requirements for this product.

Appendix A: Screen Images



Appendix B: Analysis Models

