

SOFTWARE

DESIGN

DOCUMENT

Prepared by: RohanAnilKumar

Sidharth M

Athullya R

Adithya Krishna

Contents

1. Introduction
 - 1.1 Purpose of the document
 - 1.2 Scope of the development project
 - 1.3 Definitions, Acronyms and Abbreviations
 - 1.4 References
 - 1.5 Overview
2. Design Description
 - 2.1 Assumptions
 - 2.2 Design Constraints
 - 2.3 Design Methodology
3. System Architecture Description
 - 3.1 Modules/Components
 - 3.2 Structure and relationships
 - 3.3 User Interface
 - 3.3.1 Sign up window
 - 3.3.2 Create Account window
 - 3.3.3 Data Collection window
 - 3.3.4 Routes window
4. Data Design
 - 4.1 Data Storage
 - 4.2 Data Validation
 - 4.3 Data access
 - 4.4 Data backup and recovery
5. Database Design and Schema
 - 5.1 Databases
6. Algorithm Design
 - 6.1 Problem definition
 - 6.2 Input
 - 6.3 Data Structures
 - 6.4 Pseudo-code
 - 6.5 Algorithm Complexity

- 7. Error Handling
 - 7.1 Error types
 - 7.2 Error Reporting
 - 7.3 Recovery Mechanism
 - 7.4 User feedback and support

- 8. Performance
 - 8.1 Performance Requirements
 - 8.2 Performance Testing
 - 8.3 Performance Optimization

- 9. Testing and Quality Assurance
 - 9.1 Testing Approach
 - 9.2 Techniques
 - 9.3 Test Cases
 - 9.4 Test Environment
 - 9.5 Quality Assurance

- 10. Conclusion

1. Introduction

1.1 Purpose of the document

The purpose of this software design document is to provide a clear understanding of the design, architecture, and functionality of a software that generates feasible travel paths for users within a specified budget and time duration. It serves as a blueprint for the development team and outlines the software's features, user interfaces, algorithm and database. The document also outlines user requirements and use cases to ensure that the software meets the needs of its users and provides a seamless user experience.

1.2 Scope of the development project

The scope of the development project is to create a software application that provides users with personalized travel itineraries based on their budget based on their budget and time constraints. The application will be designed to be user-friendly and efficient, with a front-end interface that allows users to input their travel preferences and generate feasible travel paths. The back-end server will process the user inputs and generate a list of recommended tourist places to visit based on the user's budget and time constraints.

The software uses algorithms and databases of routes and places to generate the travel paths. The algorithm will take into consideration various factors such as time, distance, budget. The database contains information about tourist places such as location, availability.

The software application will be developed using agile methodologies with frequent iterations and feedback from users. The development process will include several stages including requirement gathering, design, implementation, testing and deployment.

1.3: Definitions, Acronyms and Abbreviations

User: A person who is registered on the application and uses it to choose routes to travel spots.

Route: The paths generated by the algorithm from the start place to the destination.

1.4: References

1. [Algorithms for the weight constrained shortest path problem](#) by Irina Dumitrescu and Natasha Boland
2. [Effective indexing for approximate constrained shortest path on](#) by Sibor Wang, Xiaokui Xiao, Yin Yang, Wenqing Lin'

These references have been consulted for the design and development of the software application described in this document.

1.5 Overview

This software design document provides an overview of a software application that generates feasible travel paths for users who want to visit tourist places within a specific budget and time duration. The application consists of a front-end user interface and a back-end algorithmic engine. The front-end interface allows users to input their travel preferences, while the back-end algorithmic engine generates feasible travel itineraries based on various factors such as distance, travel time, available transportation modes, and tourist destinations. The software application prioritizes user experience, security, and scalability. The software application undergoes extensive testing, including functional testing, performance testing, security testing, user acceptance testing. The team also conducts frequent user feedback sessions to incorporate user suggestions and feedback into the development process.

In summary, this software application provides a seamless and personalized experience for users who want to plan their travel itineraries within a specific budget and time duration.

2. Design Description

2.1 Assumptions

- The app assumes that the route information from Google Maps API is accurate.
- The app assumes that the user will have an active internet connection to fetch route information.
- The app assumes that the user will input their starting point and destination in the form of an address or landmark, rather than latitude and longitude coordinates.

2.2 Design Constraints

Constraints in the software application that generates feasible travel paths for users based on duration and budget could include:

1. API limitations: There may be limits on the usage of the Google Maps API, such as restrictions on the number of API requests or limitations on the type of data that can be accessed.
2. Data accuracy: The accuracy and completeness of the data provided by the Google Maps API may be a constraint, which could affect the quality and feasibility of the travel plans generated by the software application.
3. Budget and duration constraints: The user's budget and duration preferences may limit the number of travel destinations, activities, and transportation options that can be included in the travel plan.
4. Availability constraints: Availability of lodging, transportation options, and activities at the desired travel destinations could limit the feasibility of the travel plan.
5. Technical constraints: Technical limitations or constraints such as processing power or memory limitations may also need to be considered during the development of the software application.

Addressing these constraints effectively will be critical to the success of the software application and to generating feasible travel plans for users based on their budget and duration preferences.

2.3 Design Methodology

The design methodology for the Trip Planner app follows an Agile approach, which is an iterative and incremental software development methodology. Agile methodologies prioritize flexibility, collaboration, and customer satisfaction, allowing for changes in requirements and feedback-driven improvements throughout the project lifecycle.

3.System Architecture Description

3.1 Modules/Components

NPM Packages used.

Google maps

The @google/maps npm package is a Node.js client library for the Google Maps Platform web services. It allows developers to easily integrate the Google Maps API into their Node.js applications to perform geocoding, directions, distance matrix, and other map-related tasks.

BCrypt

The bcrypt npm package is a popular library for hashing and encrypting passwords in Node.js applications. It uses a variation of the Blowfish encryption algorithm to generate secure password hashes that can be safely stored in a database.

Config

Node-config organizes hierarchical configurations for your app deployments.

It lets you define a set of default parameters, and extend them for different deployment environments (development, qa, staging, production, etc.).

Express

The express npm package is a popular Node.js framework for building web applications and APIs. It provides a minimalist and flexible approach to web development, allowing developers to quickly and easily create powerful server-side applications.

Helmet

Helmet by setting various HTTP headers. It provides an easy and convenient way to add security headers to HTTP responses, such as Content-Security-Policy (CSP), X-Frame-Options, X-XSS-Protection, and more.

Joi

The joi npm package is a popular Node.js library for validating and sanitizing data. It provides a simple and flexible way to define validation rules for input data and ensures that the data is in the correct format and meets certain requirements.

Jsonwebtoken

The jsonwebtoken npm package is a popular Node.js library for creating and verifying JSON web tokens (JWTs) for secure authentication and data exchange. JWTs are a compact and self-contained way of representing claims or statements about an entity, typically a user or client, in a way that can be easily transmitted between parties.

moment

The moment npm package is a popular Node.js library for parsing, validating, manipulating, and formatting dates and times. It provides a simple and flexible way to work with dates

and times in JavaScript, with a wide range of features and customization options.

Mongoose

The mongoose npm package is a popular Node.js library for interacting with MongoDB databases. It provides a simple and flexible way to define data models and query data in MongoDB, allowing developers to quickly and easily build scalable and efficient applications.

Components:


NodeJS backend, which connects to mongodb atlas using mongoose drivers. The framework express will be used for the same


ReactJS frontend which connects to the backend using REST API interface.

3.2 Structure and relationships

3.3 User Interface

3.3.1 Sign Up Window

**Trippyz**
Your complete travel partner



Sign Up

We are your destination

Name


Enter your email


Email

Enter your email


Password


Create Account





3.3.2 Create Account Window

**Trippyz**
Your complete travel partner



Welcome

We are your destination


Email


Enter your email

Password

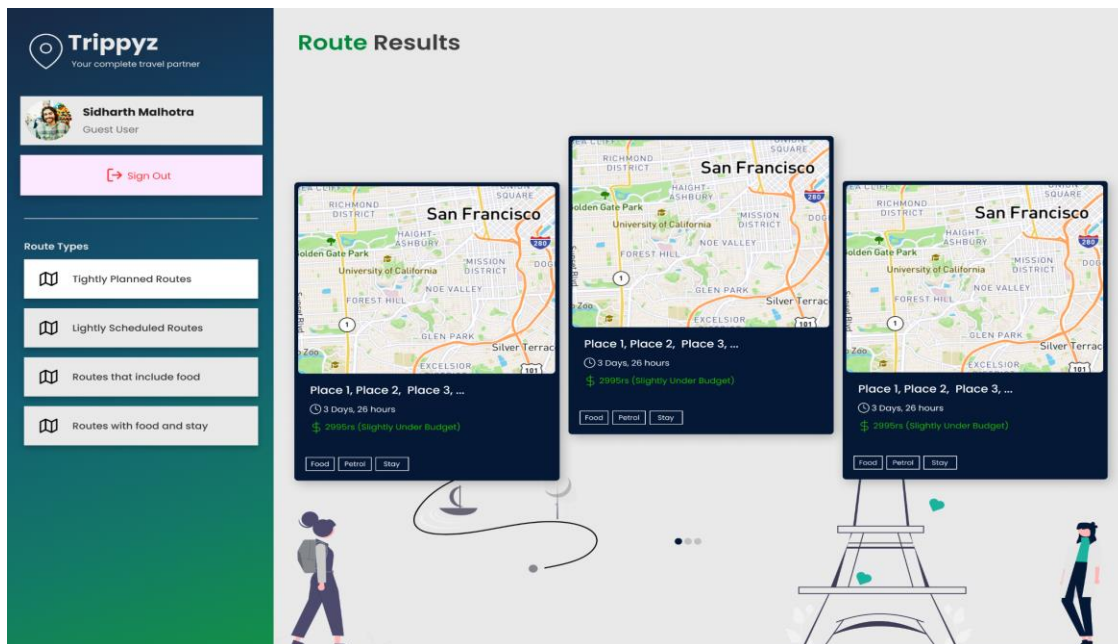
☐ Remember me

Create Account





3.3.3 Data Collection Window



3.3.4 Route Window

Trippyz
Your complete travel partner

Sidharth Malhotra
Guest User

[Sign Out](#)

Route Types

- Tightly Planned Routes
- Lightly Scheduled Routes
- Routes that include food
- Routes with food and stay

Hello There!

Let's collect some data that we need to plan your route

Where are you headed to?

Where are you starting from?

Do you have a start, end time?

Yea No

What's the duration of the trip

What's the trip start, end date

Start End

[Next >](#)

4. Data Design

4.1 Data Storage

The data storage for the software application that generates feasible travel paths for users based on duration and budget will be accomplished using MongoDB, a NoSQL document-oriented database.

4.2 Data validation

4.3 Data Access

The software application will use the Google Maps API to retrieve information about the travel destinations and transportation options. This information will be then stored in the database for future use

4.4 Data Backup and Recovery

The database will be maintained regularly by the software application to ensure that the information stored in it is up-to-date and accurate. The software will periodically check for updates to the Google Maps API and make changes to the database accordingly. The database will also be backed up regularly to prevent data loss in case of system failures.

5. Database Schema

5.1 Databases

The database schema for the software application that generates feasible travel paths for users based on duration and budget using MongoDB will consist of the following collections and fields:

Collections	Fields	Remarks
Destination Collection	_id	Unique identifier for the Document
	Name	Name of the travel Destination
	Location	Geographical location of the travel destination
Transportation Collection	_id	Unique identifier for the Document
	Type	Type of transportation
	Origin	Starting point of Transportation
	Destination	End point
	price	Price of the transportation
	duration	Duration of the transportation
User Collection	_id	Unique identifier for the document

	name	Name of the user
	email	Email of the user
Travel Plan Collection	_id	Unique identifier for the document
	user_id	Unique identifier for the user who generated the travel plan
	budget	the total budget for the travel plan
	duration	the entire duration of the travel plan
	destinations	List of travel destinations in the travel plan
	transportations	List of transportation options in the travel plan

The above schema will provide a flexible and scalable solution for storing and managing the data required for generating feasible travel paths based on duration and budget preferences of the users.

6. Algorithm

6.1 Problem definition

The problem definition for the algorithm used to generate feasible travel paths for users based on duration and budget is as follows:

Given a list of travel destinations, transportation options, and the user's budget and duration preferences, generate a customized travel plan that meets the user's requirements. The travel plans should include a list of travel destinations, transportation options, and activities that fit within the specified budget and duration constraints.

The algorithm should take into account the following factors while generating the travel plan:

1. Distance and travel time: The algorithm should consider the distance between the travel destinations and the travel time required to reach them. This will help ensure that the travel plan is feasible within the user's specified duration.
2. Cost: The algorithm should consider the cost of transportation, lodging expenses while generating the travel plan. This will help ensure that the travel plan is feasible within the user's specified budget.

By taking into account the above factors, the algorithm will generate a feasible and customized travel plan that meets the user's requirements.

6.2 Input

The current inputs required by the algorithm to create the paths are Starting place start, destination end, budget, graph of the area Containing all the possible paths from start to end. The graph is fetched from google Maps API.

6.3 Data Structures

The algorithm uses several data structures to perform its operations efficiently. The data structures commonly used in the CCSP algorithm:

1. Graph Data Structure: The CCSP algorithm operates on an undirected graph, which is typically represented using an adjacency list or adjacency matrix. An adjacency list is a data structure that stores the neighbors of each vertex in a linked list or array, while an adjacency matrix is a 2D array that stores the edges between vertices.
2. Disjoint-Set Data Structure: The CCSP algorithm uses a disjoint-set data structure to keep track of the connected components in the graph. A disjoint-set data structure is a collection of disjoint sets, where each set represents a

connected component. The data structure provides efficient operations to union two sets and find the root of a set.

3. Stack Data Structure: The CCSP algorithm uses a stack data structure to keep track of the subgraphs that have been processed and the order in which they were processed. The stack is used to backtrack when a dead-end is reached in the search space.
4. Map Data Structure: The CCSP algorithm uses a map data structure to keep track of the subgraphs that have already been processed and their corresponding permutations. This allows the algorithm to avoid revisiting the same subgraph multiple times.

These data structures allow the CCSP algorithm to efficiently search the space of all possible permutations of the connected component subgraphs in an undirected graph.

6.4 Pseudo-code

CCSP Algorithm:

```
import heapq

def ccsp (graph, start, end, budget):

    distances = {node: float('inf') for node in graph}

    distances[start] = 0

    queue = [(0, start, 0)]

    while queue:

        current_cost, current_node, current_distance =

            heapq.heappop (queue)

        if current_node == end:

            return current_distance

        if current_cost > budget:
```



```
        continue

    for neighbor, weight, cost in graph[current_node].items():
        distance = current_distance + weight
        total_cost = current_cost + cost
        if total_cost <= budget and distance < distances[neighbor]:
            distances[neighbor] = distance
            heapq.heappush(queue, (total_cost, neighbor, distance))

    return None
```

6.5 Algorithm Complexity

The Computational Complexity of the Connected Component Subgraph Permutation (CCSP) algorithm depends on the size of the input graph and the number of connected components in the Graph.

The CCSP algorithm can be used to find all possible permutations of the connected component subgraphs in an undirected graph. The time and space complexity of the CCSP algorithm are both exponential in the number of connected components in the graph.

7. Error Handling

- 7.1 Error types
- 7.2 Error Reporting
- 7.3 Recovery Mechanism
- 7.4 User feedback and support

8. Performance

8.1 Performance Requirements

8.2 Performance Testing

8.3 Performance Optimization

9. Testing and Quality Assurance

9.1 Testing Approach

9.2 Techniques

9.3 Test Cases

9.4 Test Environment

9.5 Quality Assurance

10. Conclusion