Electronic Tour Planner

(ETP)

Graduation Project
Computer Science Department
FCI, LUXOR

Project Advisors:

Dr. Mohamed Ramadan

Dr. Amani Salama

Submitted by

Mohamed Fathy Mohamed

Gemian Talasoun Sadeq

Mohamed Gamal Negm

Hassaan Nabil Hassan

2019 – 2020

ABSTRACT

The electronic tour planner (ETP) system generates plans for tourists who would like to visit Luxor. The ETP is a mobile application and has three types of data, user data (favorite list, forbidden list, preferences), trip data (start time, end time, budget) and places data (name, open time, close time, preferences, minimum duration, minimum cost, description). Most of these data classified into hard constraints such as (budget, trip start and end times, …etc) and soft constraints such as (favorite list, preferences, …etc). ETP uses the hard constrains to generate candidate plans and uses the soft constrains to evaluate the generated plans. The planner system is divided into three parts and each part sends his output data to the next part: First part responsible for getting all data; Second part preparing the data; Third part responsible for getting the user's plan. The ETP used the tabu search algorithm to generate his plan and used (swap, insert, delete) operations to get the neighbors. Also, the algorithm divides the plan into 2 parts morning and night parts to do his operations on each part sequentially. And it divides all places into the morning, night, and full-time places based on their working times. We did more than 70 tests in generating plans from 20 places in Luxor with different plan's constraints and these tests are explained in the functional test phase. From these tests the maximum time needed to generate the plan is 3 seconds. Also, the application provides more services like Ekko chatbot that used to improve the user usability by helping him to use the application, The dashboard which is a store of shared plans which used to allow more than one user use the same plan or exchanging the opinions between users to get a better plan from an old shared plan, The places information which allows the user to know information about the places in the system.

Keywords: Electronic tour planner (ETP), point of interests (POIs).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ETP	***Electronic Tour Planner***
POIs	***Point of Interests***
TTDP	***Tourist Trip Design Problem***
TOPTW	***Team Orienteering Problem with Time Windows***
MVC	***Model View Controller***
Ekko	***Assistant Chatbot***
CF24T12	***Convert Time From 24 To 12***
CF12T24	***Convert Time From 12 To 24***
SFW	***Satisfaction Factor Weight***
SFs	***Satisfaction Factors***
TTW	***Travel Time Weight***
TT	***Travel Time***
WTW	***Waste Time Weight***
WT	***Waste Time***
MT	***Maximum Tries***

1. INTRODUCTION

Tourism is one of the most important way of increasing income for several cities such as Luxor. Therefore, some governments are using technology to improve in this industry. For this reason, ETP was created to contribute to facilitate the tourism process. Doing the plan of visit that includes most interesting Points of Interests to visit, for the available time, is usually a complex task. In such situations, it would be helpful for the tourist to have a system that runs on a hand held device, which would enable him to automatically plan the touristic trip. In general, systems like that tend to fulfill as much as possible the satisfaction of tourist by making a personalized trip plan.

The problem is generally known as the Tourist Trip Design Problem (TTDP) which is a route-planning problem on multiple Points of Interest (POIs). TTDP can be considered as an extension of the classical problem of Team Orienteering Problem with Time Windows (TOPTW).

A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms. Notable examples of metaheuristics include genetic/evolutionary algorithms, tabu search, simulated annealing, variable neighborhood search, (adaptive) large neighborhood search, and ant colony optimization, although many more exist.

Tabu search, created by Fred W. Glover in 1986 and formalized in 1989, is a metaheuristic search method employing local search methods used for mathematical optimization. Local (neighborhood) searches take a potential solution to a problem and check its immediate neighbors (that is, solutions that are similar except for very few minor details) in the hope of finding an improved solution. Local search methods have a tendency to become stuck in suboptimal regions or on plateaus where many solutions are equally fit. Tabu search enhances the performance of local search by relaxing its basic rule. First, at each step worsening moves can be accepted if no improving move is available (like when the search is stuck at a strict local minimum). In addition, prohibitions (henceforth the term tabu) are introduced to discourage the search from coming back to previously-visited solutions. The implementation of tabu search uses memory structures that describe the visited solutions or user-provided sets of rules. If a potential solution has been previously visited within a certain short-term period or

if it has violated a rule, it is marked as "tabu" (forbidden) so that the algorithm does not consider that possibility repeatedly.

ETP makes this personalized trip plan via three parts first part (tourist input data) these data classified to hard constrains like (budget, start and end time) and soft constrains like (preferences, forbidden places and favorite places), second part (prepare) this part responsible for preparing data such as filtering POIs and make the initial plan, third part (Algorithm) responsible for getting the user's plan and showing it to the user. The user can modify this plan by adding or removing places. The ETP has an assistant and it is a chatbot helps users to use our system by easily way, and the ETP has the dashboard which is a store of shared plans which used to allow more than one user use the same plan or exchanging the opinions between users to get a better plan from an old shared plan, the places' information which allows the user to know information about the places in the system.

2. ANALYSIS AND DESIGN

2.1 DOMAIN ANALYSIS

The problem of trip planning has received wide concerns in recent years. More and more people require the service of automatically confirming the optimal tour route. When users assign the source and the destination, and the time limit of the tour, how can automatically make schedule help user to visit point of intetrets Correspond to his prefrances.

The application, TripBuddy[1], was developed to learn user's behavior based on empiric data, which used to offer relevant destinations to certain user by using KMeans Clustering. TripBuddy is application which suggests optimal route with detail information of destinations, schedule, cost and duration to ease user's travel plan and it also gives recommendation based on user's browsing behavior; System plan it is not good because user must choice places he wants to visit and system choice these places according short path location for some places.

This paper is concerned with developing a fast algorithm [2] that can be embedded within a (real-time) personalized tour planning recommender system for use by website/mobile apps. they introduce an extended mathematical model for Tourist Trip Design Problem that includes some additional constraints, such as different start and end nodes and the time budget for each route. We then propose an Iterated Local Search (ILS) approach to solve Tourist Trip Design Problem.; Do

not use human behavior (rate place) in his constraint to make schedule and we think it is very important to add in constraints because it makes schedule beater than it without Browsing behavior. people can't modify recommended plans (i.e., insert/remove/replace/reorder the POIs in the plans). The System just make one plan. the individual user's profiles, preferences, travel planning result and feedbacks are also recorded and stored in the database. The information obtained from travel planning result is used to improve the popularity of all locations and gain more suitable recommendations.

CT Planner v4[3] Enables the user to design a tour plan with the system in a collaborative manner. Application mainly targets foreigners and is expected to stimulate their hidden/unattend needs of plan consultation. people can modify recommended plans (i.e., insert/remove/replace/reorder the POIs in the plans). Shows three items for user: tour condition, user profile, and command bottoms. The tour condition consists of five items: duration, start time, day of the week, walking speed, and reluctance to walk. If you modify the tour condition, your plan is revised promptly. For instance, if you set the start time to 5:00pm, your plan will skip most museums because they are already closed. Similarly, if you set the reluctance to walk to yes, the walking distance of your plan will become shorter. If you click a POI's name on the map or the agenda, a small info-window appears at its location. The window also shows several buttons. Once you click Visit button, the system generates the plans which visit this POI as long as possible. Finally, +10/-10 button allows you to adjust the staying time.; when user design a plan, the system select place randomly and by rate. The system does not set the plan by budget.

User-adapted travel planning system [4] study proposes a personalized travel planning system (PTPS) that possesses a time framework (TF) concept with adjustable recommendation results to resolve the mentioned problems. Operating the PTPS involves users imputing their personal requirements through an Internet platform, such as the number of travel days and accommodation or hotel budgets. Through a travel planning module (TPM), a preliminary traveling schedule is constructed, and adjustment functions assist user to get a satisfied travel plan. The TPM employs a schedule reasoning method (SRM) to plan a travel schedule based on TF. A collocation of the travel requirement match module cross-references the user requirements and travel destination to select the destination that best satisfies the user's requirements. Locations that fulfill the user's requirements are connected and formatted within the TF based on time to plan a preliminary travel schedule. Regarding the interface design, this system provides a travel schedule replacement design with the following three location categories of replacement options: attractions (A), dining and restaurants (R), and accommodations and hotels (H). The PTPS arranges the different location categories items based on popularity,

enabling the user to replace unsatisfying items in the travel schedule using an adjustable interface and to obtain a schedule that matches their requirements. The travel schedule item that is selected by the user is recorded in the database through a feedback mechanism, serving as an indicator for calculating popularity. This increases the accuracy rate and enables future plans to more closely match the user's requirements.; Begin by using c1 as the point of departure, and search for the locations that match the user requirements (M) within the search area (b). The location default only contains three categories: A, R, and H. If a location matches the user's requirements and the time framework block category, it becomes a candidate location for the next point of departure c2. If multiple candidate locations exist, the most popular location is used as the point of departure. If no locations match the user's requirements, the location category of the time block is considered and the most popular location in that category is used as the next point of departure (c2).

2.2 FUNCTIONAL REQUIREMENTS

- **Input data:** User enters some data that system needed such as start time and end time to his plan, budget, preferences, forbidden places and favorite places.
- **Get data:** Getting information about POIs like opening and closing time, cost and minimum travel time between places.
- **Make plan:** Based on the user's information and the user's plan constrains planner system generate a plan.
- **Display plan**: Show the plan to the user.
- **Modify**: User can add places to his plan or can delete places from it.
- **Ekko assistant:** It is like a chatbot, user can use it to help him how use our system.
- **Share**: User can share his plan to the dashboard to be shown by other users and they can save it in their lists.

2.3 NON-FUNCTIONAL REQUIREMENTS

- ***Usability***: A pillar of the interactive application, usability gives a much better experience for the user and simplifies the task of entering and submitting the correct information that matters to the system in the correct format.
- ***Response Time***: In addition to the benefits a good time performance presents in enhancing the overall user experience, it is primordial for the system to send back results in a reasonable amount of time.

- ***Scalability***: when system make plan, can user make modify on it from his point of view and his need.
- ***Accuracy***: If the system is to be of any use in the real world, the schedules that are sent to the user as solutions to his/her queries must be accurate.
- ***Robustness***: Because user input could be erroneous, the application must make sure that a user request has the correct parameters and that their values are valid. Therefore, the application -and the system as a whole- must handle and respond to erroneous parameters appropriately in order to avoid unexpected failures.

2.4 RISK / CONSTRAINTS

- Filter the types of POIs according to user preferences.
- Define the constraints that would use to generate an efficient schedule.
- Develop an algorithm that builds an optimal schedule.
- Assess the performance and scalability of the algorithm that make schedule when use with it another algorithm like algorithm to make filter and compute short path.
- **Performance risk**, the risk that the project will fail to produce results consistent with project specifications.
- **Schedule risk**, the risk that activities will take longer than expected. Slippages in schedule typically increase costs and, also, delay the receipt of project benefits, with a possible loss of competitive advantage.
- **Cost risk**, typically escalation of project costs due to poor cost estimating accuracy and scope creep.
- **Strategic risks** result from errors in strategy, such as choosing a technology that can’t be made to work.

2.5 PROJECT PLAN

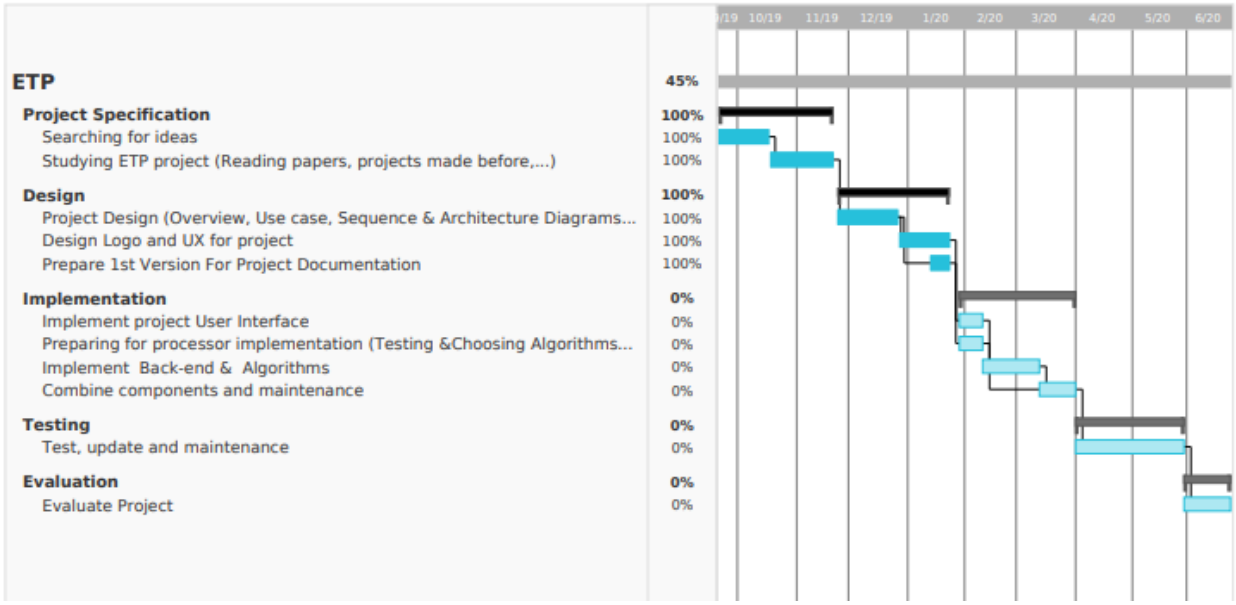


FIGURE 1. Project Plan

2.6 USE CASE REQUIREMENTS

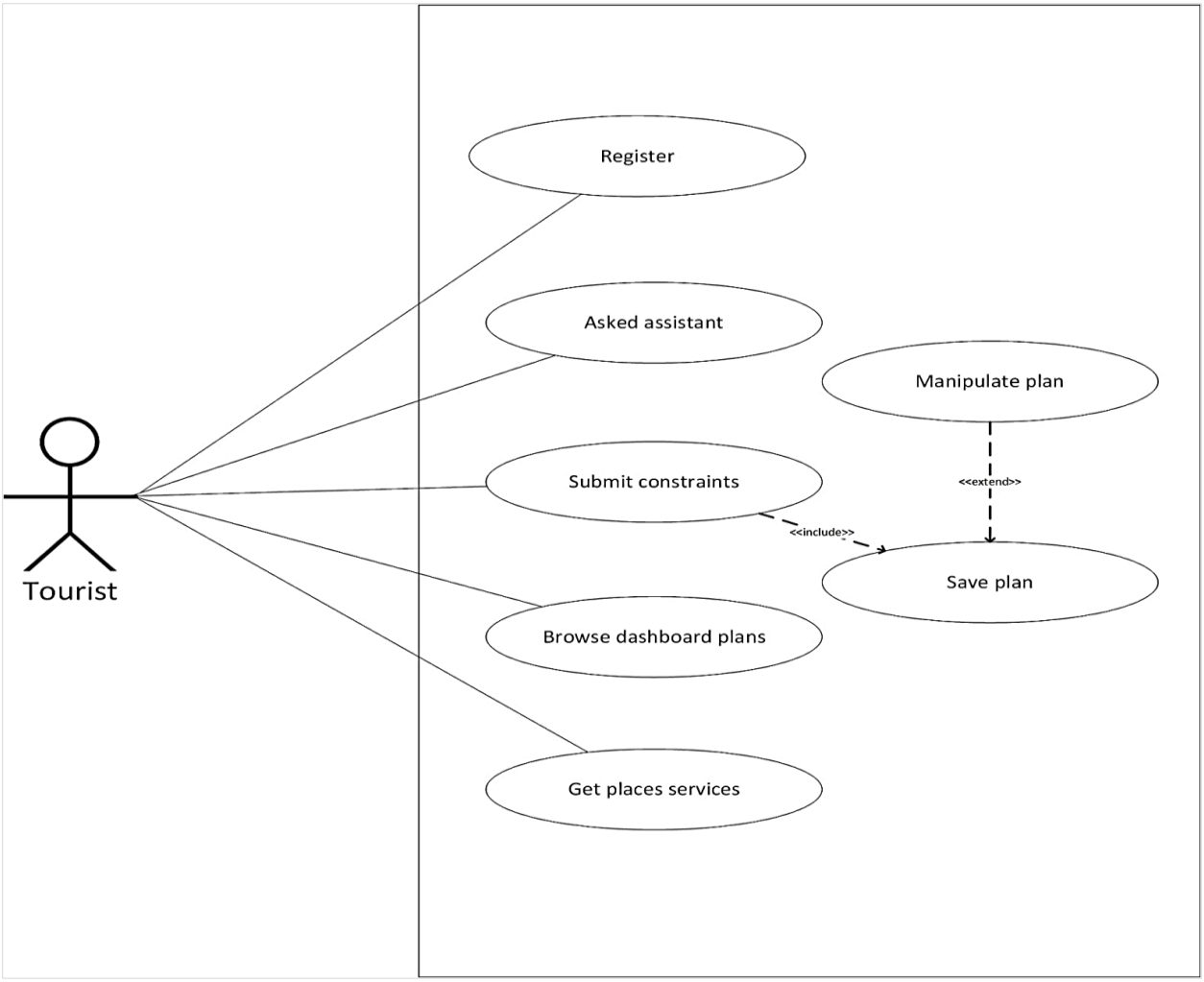


Figure 2 Use Case Requirements.

Use case ID	ETP_1	
Use case name	Register user	
Actor	Tourist	
Description	The tourist will register using name, Password, and email to register in the system.	
Steps Performed		**Information for steps**
1. User enters information in registration page and clicks submit.		Name, password, email, and preference.
2. Checked validation of information.		Registration page.
3. Registration confirmation page is displayed to confirm registration information.		Confirmation.
4. Add user to database.		Confirmation.
preconditions	The tourist given their account details to access the system.	
post conditions	Tourist can be access to the system.	
Success guarantee	The user image is at the header of every page in application	
Requirements	Allows the tourist access to system.	
priority	High.	
risk	Medium.	

Table 1 Register Use Case Scenario

Use case ID	ETP_2	
Use case name	Ask assistant	
Actor	Tourist	
Description	The tourist ask assistant using ekko chatbot.	
Steps Performed		**Information for steps**
1. User ask assistant about some services.		About ETP, Ekko services.
2. Checked which service the user need about ETP.		Profile, dashboard, places information, home, constraints, review plan.
3. Checked which service the user need from services.		Time between two places, place information.
preconditions	Ekko fulfills the user request.	
post conditions	Explanation of the user's request is displayed.	
Success guarantee	The fulfillment of the user request and how to use the application.	
Requirements	The fulfillment of the user request.	

priority	High.
risk	Medium.

Table 2 Ask assistant Use Case Scenario

Use case ID	ETP_3	
Use case name	Submit constraints.	
Actor	Tourist	
Description	Tourist send constraints to make schedule.	
Steps Performed		**Information for steps**
1. System get constraints that tourist send it.		Start time and end time, budget, preferences, forbidden places, favorite places.
2. System display POIs that user will be go to.		Opening, closing time for each POIs and cost.
3. User can be adding another POI.		Opening, closing time for each POIs.
preconditions	Tourist has already had account and register on it, input constraints and make request to plan.	
post conditions	Tourist can make modify on plan.	
Success guarantee	System make schedule and view the plan.	
Requirements	The system allows the tourist make plan.	
priority	High	
risk	medium	

Table 3 Submit constraints Use Case Scenario

Use case ID	ETP_4	
Use case name	Browse dashboard plans	
Actor	Tourist	
Description	The tourist browses the plans implemented by other users.	
Steps Performed		**Information for steps**
1. System browse plans that user share it.		Opening, closing for each POIs and cost.
2. User can be use plan that another user makes it.		
3. Search for a place in more than one schedule and view these schedules.		Using place name.

4. User can write a comment to inquire about some things.	Comment.
preconditions	The tourist has a schedule showing the places another tourists goes to, showing the time of entry and exit of each place.
post conditions	Tourist can be browse all plans shared.
Success guarantee	System all plans shared.
Requirements	The system allows the tourists share them plans.
priority	High
risk	medium

Table 4 Browse dashboard plans Use Case Scenario

Use case ID	ETP_5
Use case name	Save plan
Actor	Tourist
Description	The tourist save the plan to browse places he go it.
Steps Performed	**Information for steps**
1. System browse the tourist plan.	Name plan, cost, place, start time, and end time.
2. User can be share his plan to other users.	
preconditions	The tourist has a schedule showing the places he goes to, showing the time of entry and exit of each place.
post conditions	Tourist can be browse his plan.
Success guarantee	System view the schedule.
Requirements	The system allows the tourist view his plan.
priority	High
risk	medium

Table 5 Save plan Use Case Scenario

Use case ID	ETP_6
Use case name	Get places services
Actor	Tourist
Description	Tourist can be show all places to go.
Steps Performed	**Information for steps**
1. System show all places to go.	Place name.

2. User can be show details about each place.	
3. User can be adding places to forbidden and favorite places.	
preconditions	The tourist can know information about place.
post conditions	Tourist can be adding this place to forbidden or favorite places.
Success guarantee	The system makes it easy for the user to know the location and its details.
Requirements	The system helps the user if the place is suitable for him or not.
priority	High
risk	medium

Table 6 Get place services Use Case Scenario

Use case ID	ETP_7
Use case name	Manipulate plan
Actor	Tourist
Description	Tourist can modify on the plan before save it.
Steps Performed	**Information for steps**
1. System browse the tourist plan.	Name plan, cost, place, start time, and end time.
2. User can be remove the place he doesn't want.	
3. User can be adding places to plan.	Start, end time, place name, and place number.
preconditions	The tourist can add the place he wants and be remove the place he doesn't want.
post conditions	Tourist reached the plan he wanted.
Success guarantee	It is possible that the tourist reduces the number of places in the plan.
Requirements	The system allows the user to choose the appropriate plan for him.
priority	High
risk	Medium

Table 7 Manipulate plan Use Case Scenario

2.7 DESIGN CLASSES

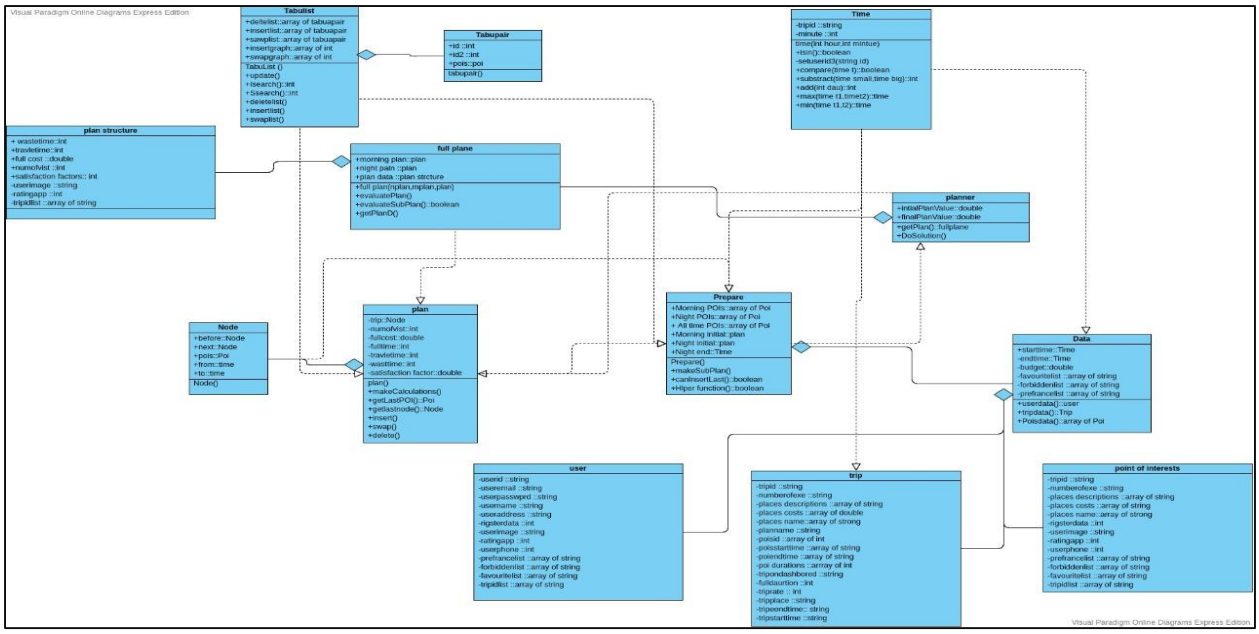


Figure 3 Design Classes

2.8 SEQUENCE DIAGRAM

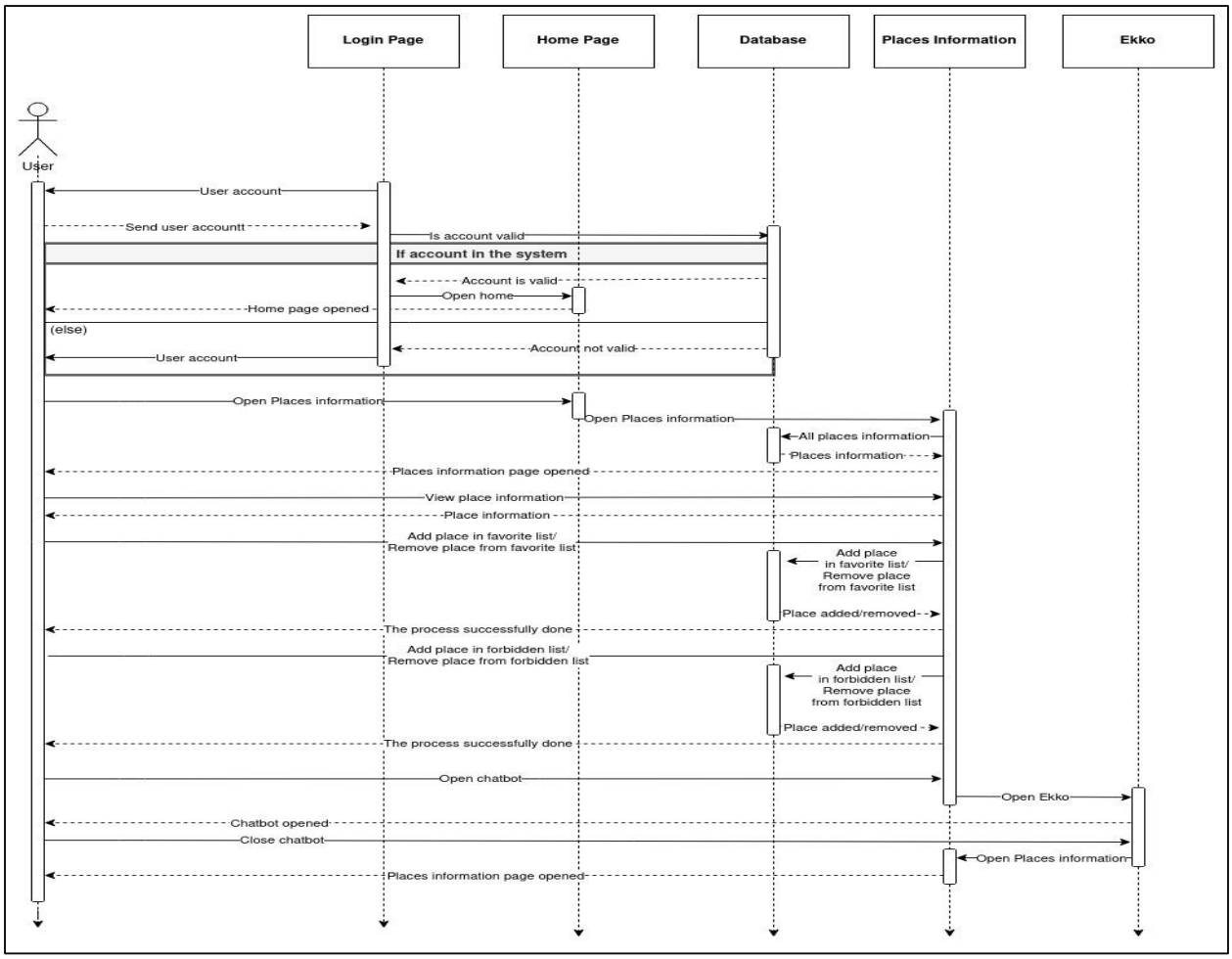


Figure 4 Places information sequence diagram

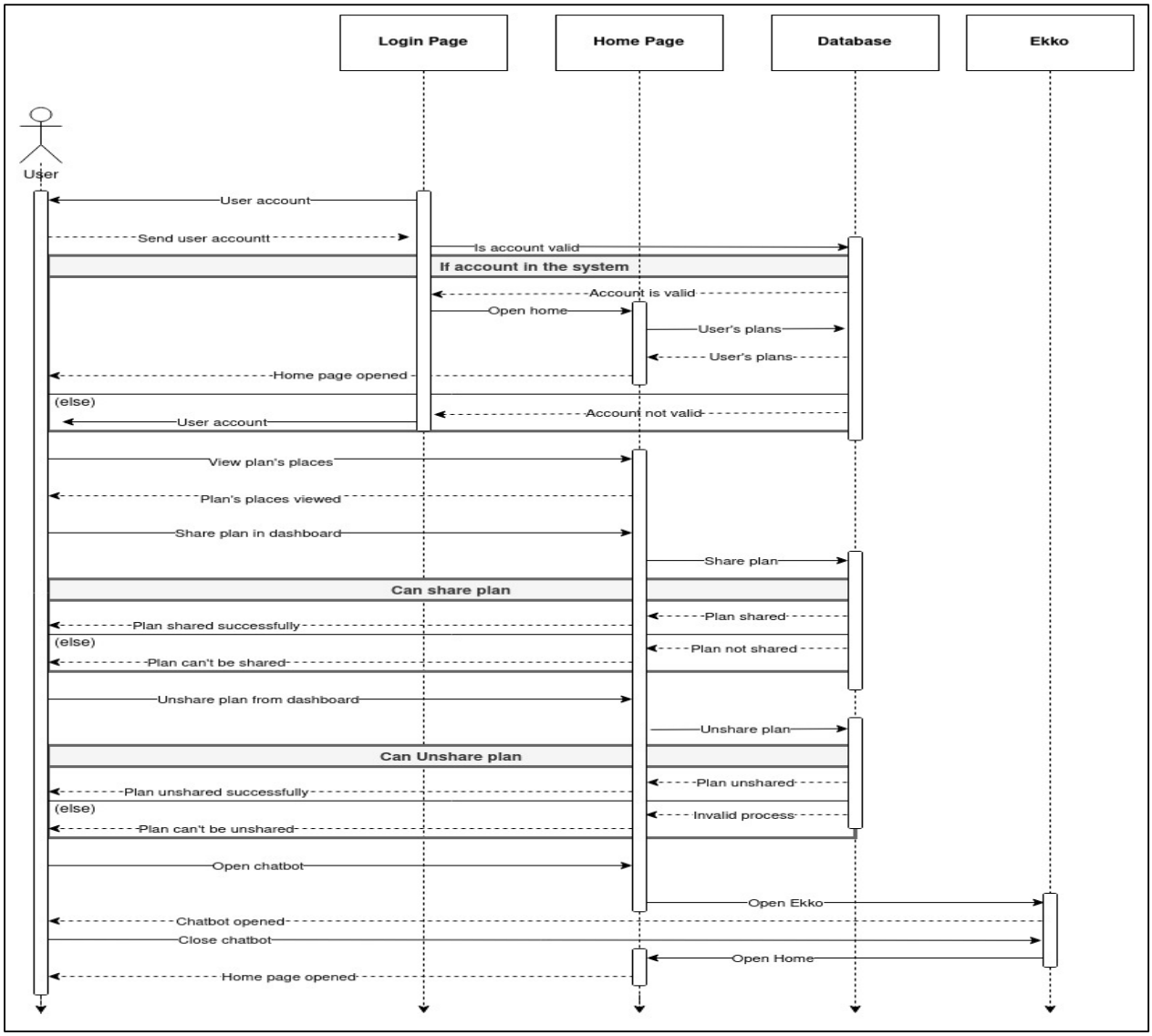


Figure 6 Home sequence diagram

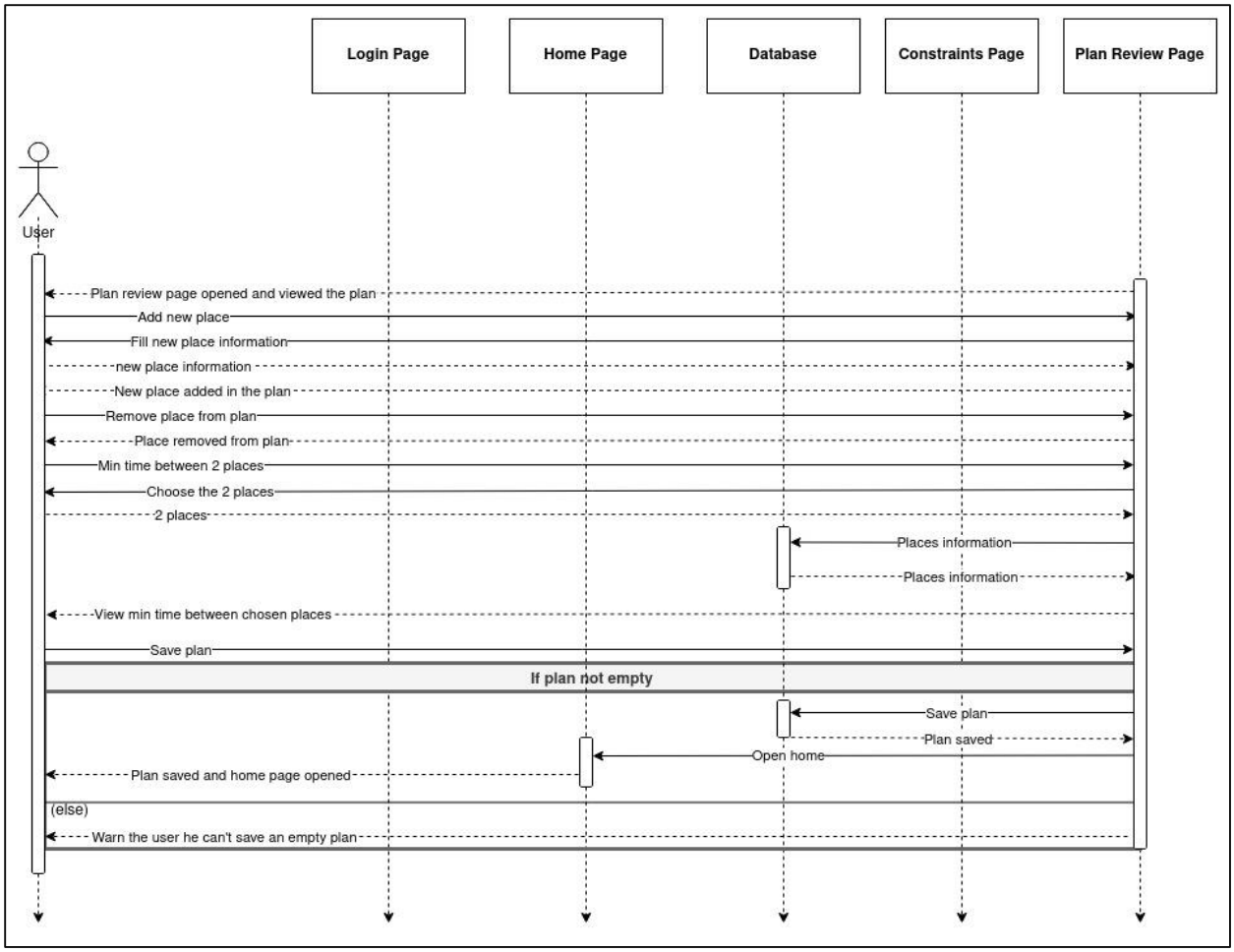


Figure 7 Modify plan sequence diagram

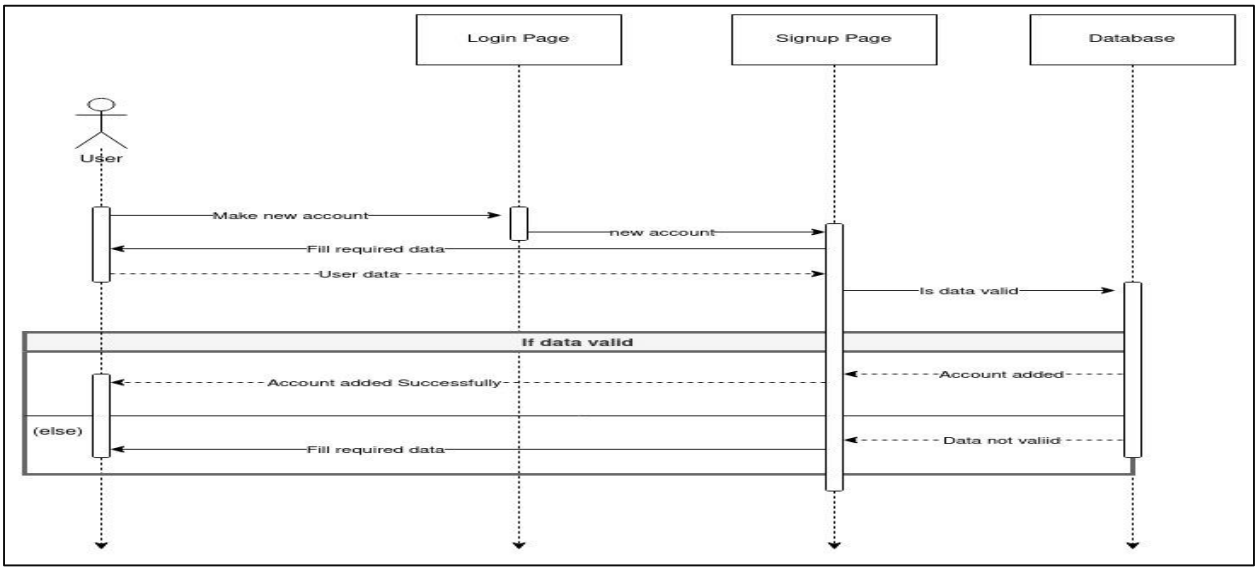


Figure 5 Sign-up sequence diagram

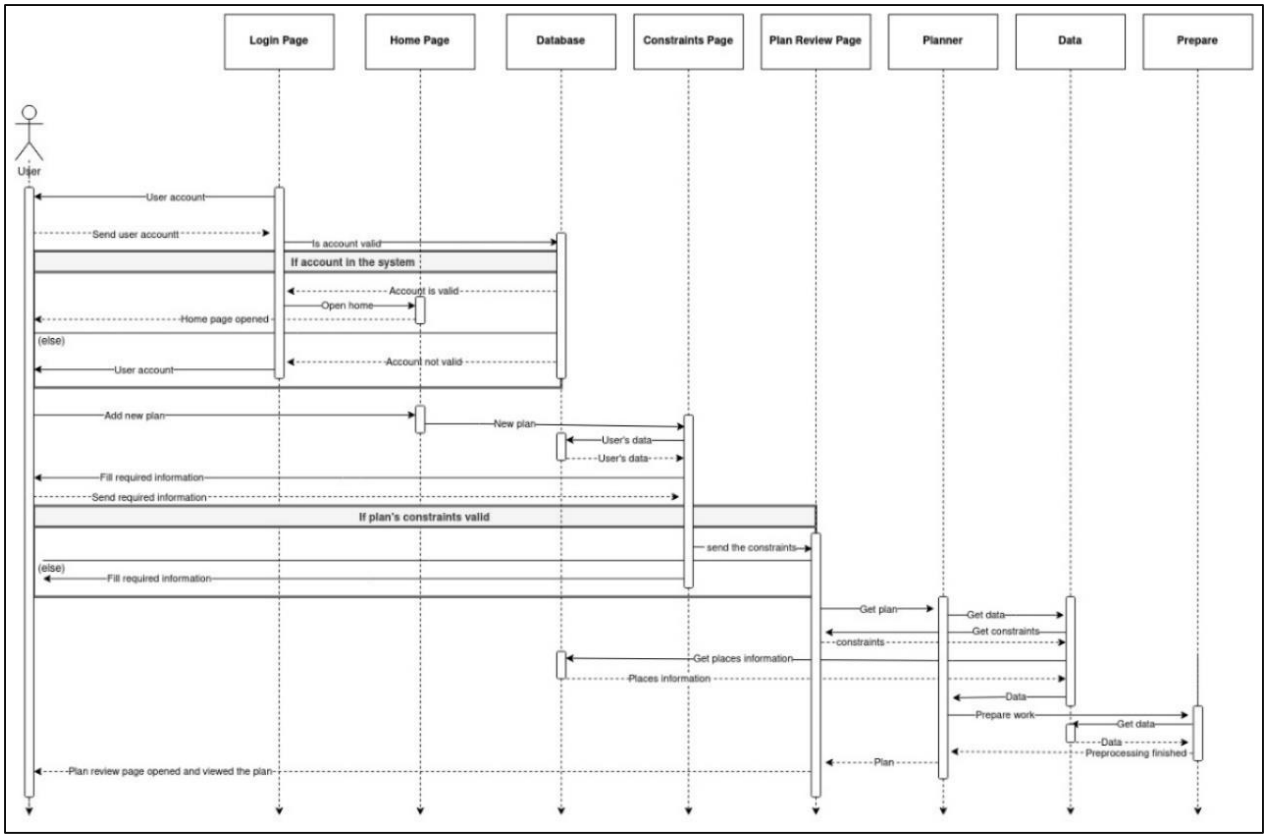


Figure 8 add plan sequence diagram

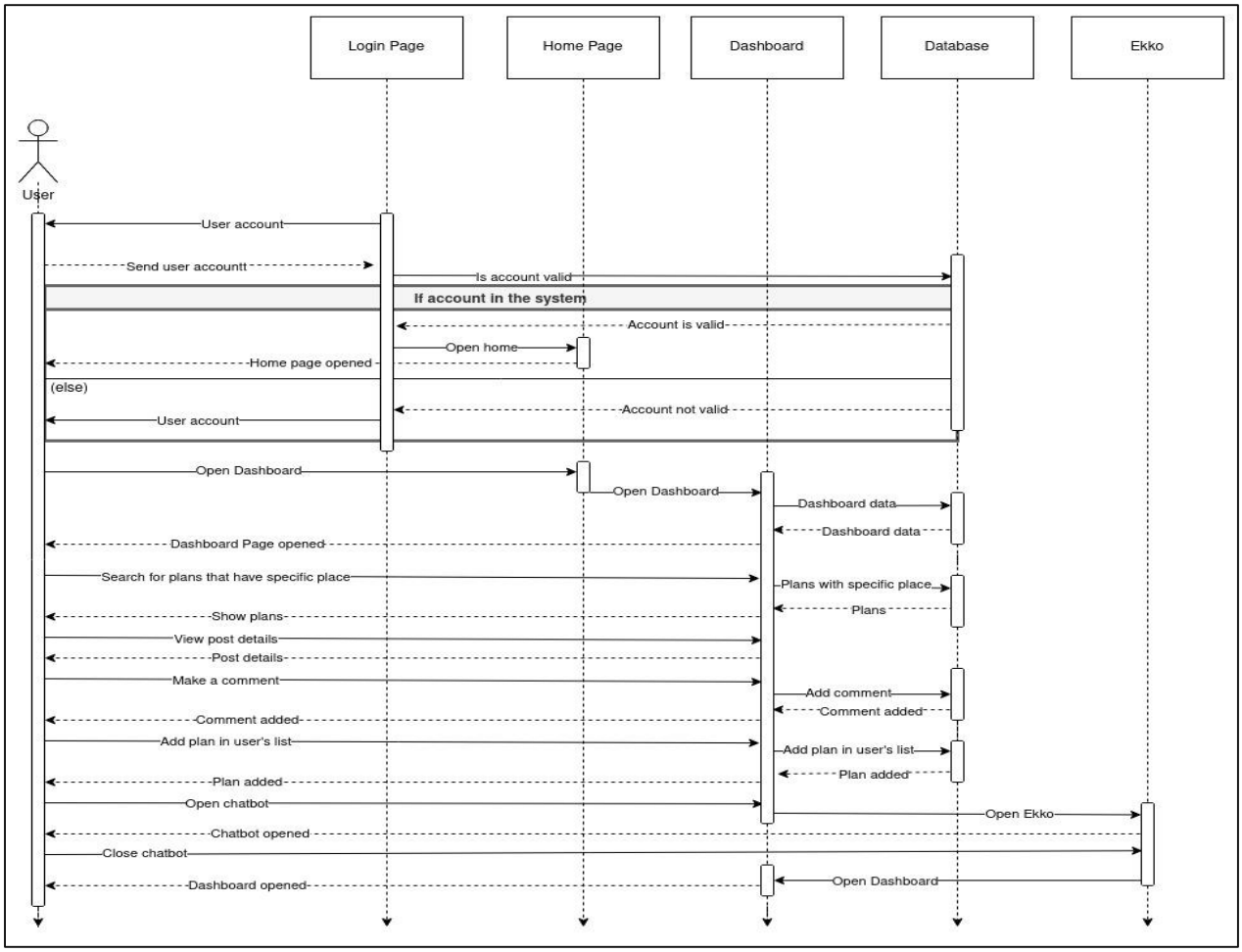
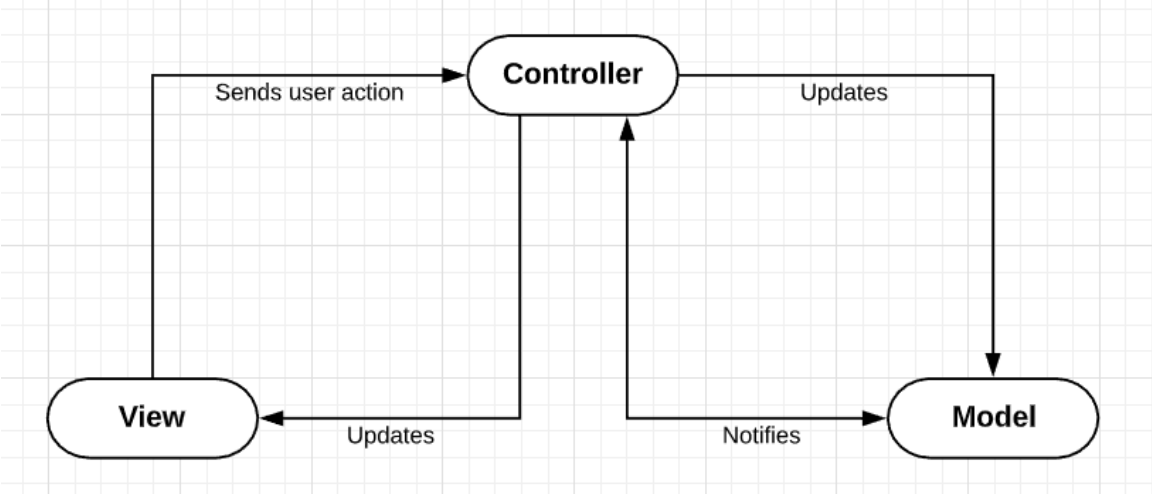


Figure 9 Dashboard Sequence diagram

2.9 SOFTWARE ARCHITECTURE

In this section will present electronic tour planner architecture pattern.

- Using MVC Architecture:
  MVC architecture is an architectural pattern that separates an application into three main logical components: the model is responsible for the data or a data access layer, the view is responsible for the graphic display of data, and the controller serves as the glue between the View and the Model. The Controller doesn't play a mediator role between the View and Model. It alters the Model based on the user's activity on the View, as well as updates the changes with the View. Each of these components are built to handle specific development aspects of an application. there is no direct connection between the View & the Model and the Controller serves as a mediator



between them as detailed below:

Figure 10 MVC Architecture

The Controller sends the message to the Model based on the incoming request invoked by the View. Model communicates the changes in the state to the controller and the controller updates the changes to the View object. The communication between the Model and View are indirect through the controller. The View doesn't bother about how the actions are performed but maintain GUI representation and delegates application-specific decisions of the interface behavior to the controller.

The controller class implements the interface of an application controller, which performs actions based on the user's request like saving data, entering data, canceling an action, etc. The sequence diagram explains the transfers in ETP's MVC architecture:

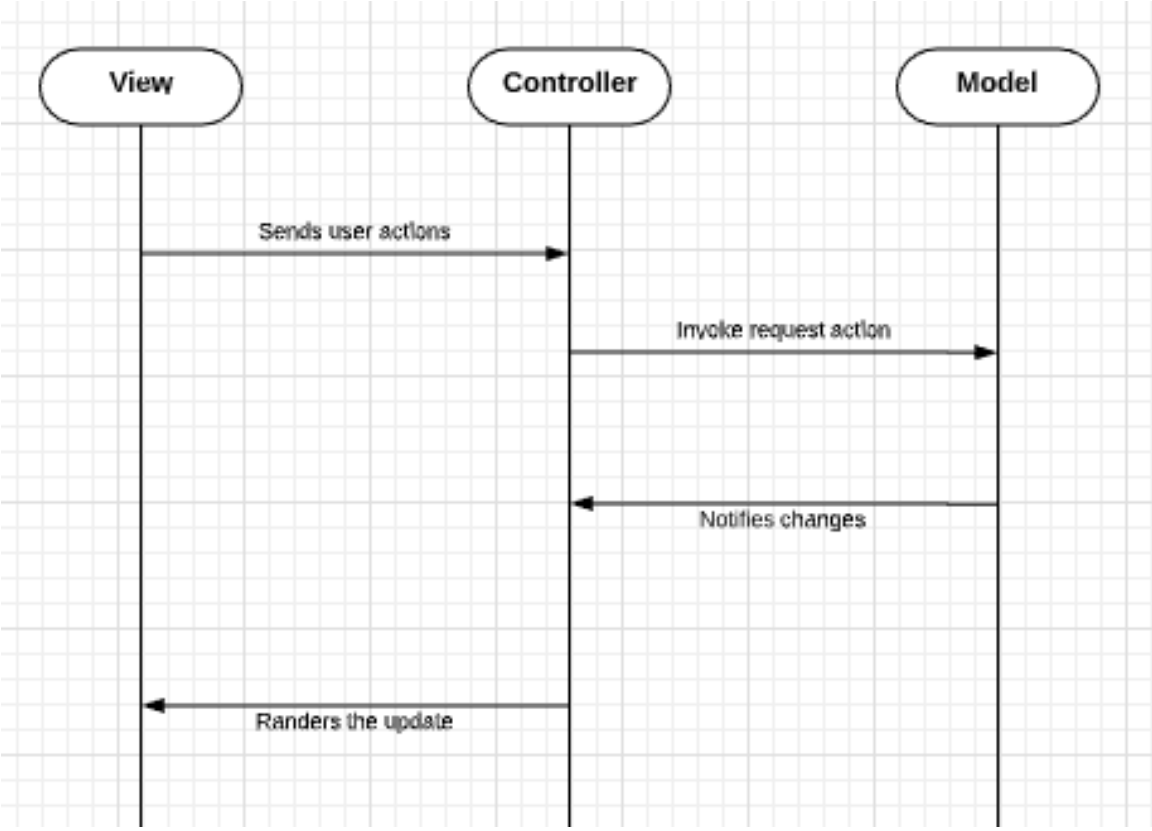


Figure 11 MVC Architecture Transfers

Note In this design pattern, there is no direct connection between the Model and the View. In addition, the controller includes the process logic of representation.

The Electronic Tour Planner system is composed of several modules at the level of the backend, as shown in Figure Each component is listed and its role described in this section.

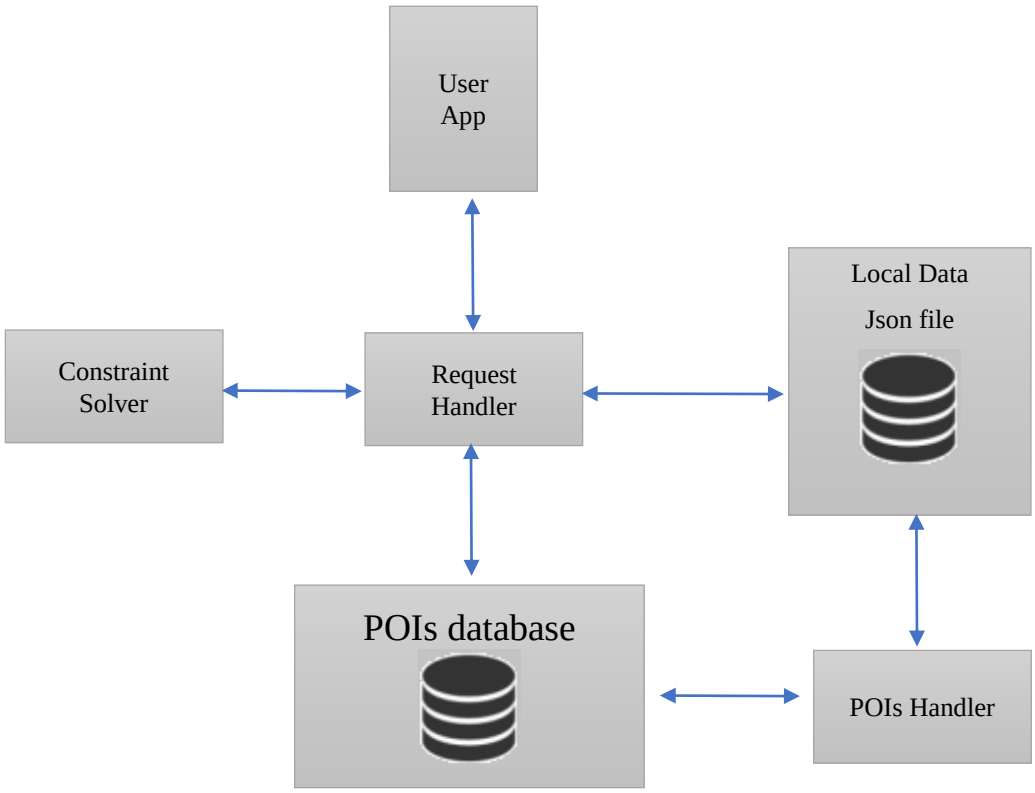


Figure 12: System architecture overview

- **Request Handler**

  The main component of the backend is the Request Handler. It receives requests from the application and manages the whole process behind generating and sending back responses. The Request Handler responds to two types of requests from the Android application; it sends the list of POIs that should appear in the schedule when the application starts, and processes schedule requests. In order to achieve these two objectives, it relies on several other components, which will be introduced in the remaining parts of this section.

- **Constraint Solver**

  The Constraint Solver represents the core logic of the backend. Its role is to make decisions related to scheduling and travel planning based on the parameters that were specified by the user in a schedule request. Therefore, the Constraint Solver has to consider the respective opening and closing

times of all POIs, the time and distance between every pair of POIs, in addition to the travel period of the tourist. Generating the best schedule means that the result should cover all of the POIs while minimizing the time and distance costs.

- **POIs Handler**

  The POIs Handler is in charge of keeping track of new places that are scheduled to take place in the city. It communicates with several external data sources and updates the list of upcoming places whenever it finds new entries. The POIs Handler is called periodically to keep the list up-to-date.

- **POIs Database**

  The Point of Interests (POIs) Database is consists of a geospatial database of Luxor cite, which contains data about its landmarks and different types of paths and routes. In this project, the POIs Database is used as a source of data where the Request Handler finds the shortest route between two POIs, which allows it to store the route as a candidate on one hand, and use the route's time and distance to build the data matrix that is sent to the Constraint Solver on the other.

2.10 USER INTERFACE MOCKUP

Figure 62: Splash Screen

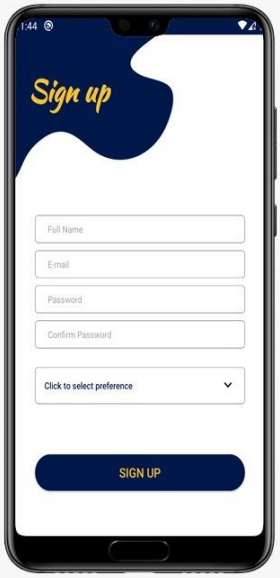


Figure 63: Sign Up Screen

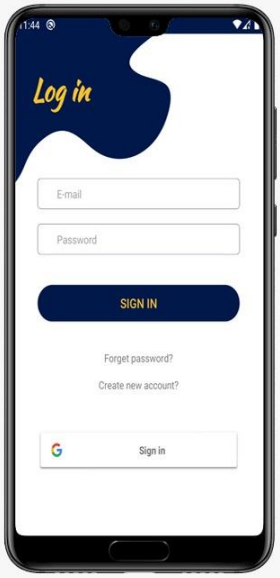


Figure 64: Login Screen

Figure 65: Home Screen

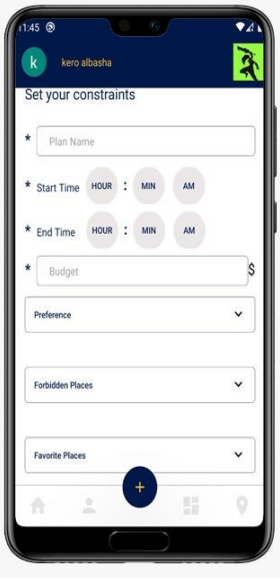


Figure 66: Add Plan Screen

Figure 67: Profile Screen

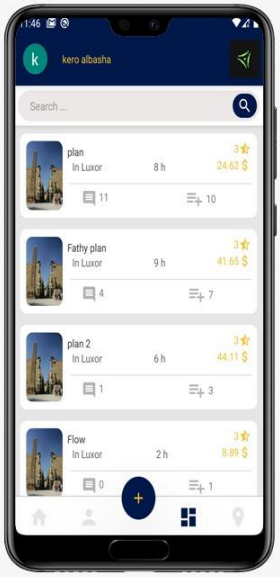


Figure 68: Dashboard Screen

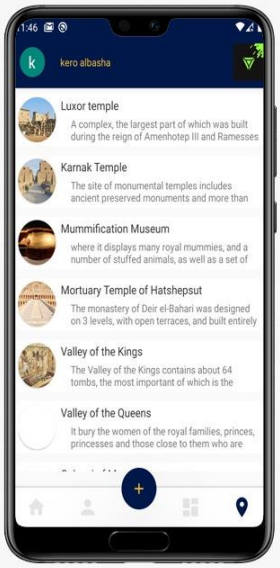


Figure 69: Places Screen

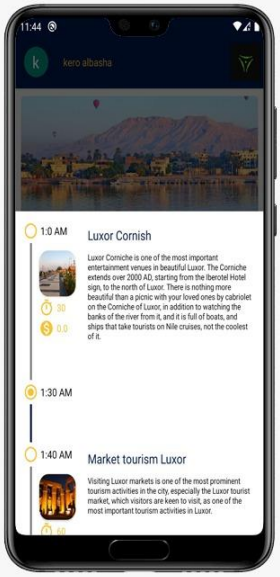


Figure 70: Show Plan Screen

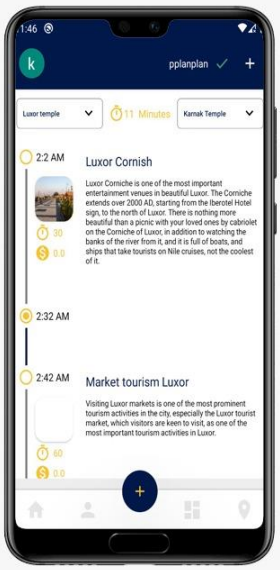


Figure 71: Apply plan Screen

Figure 72: Settings Screen

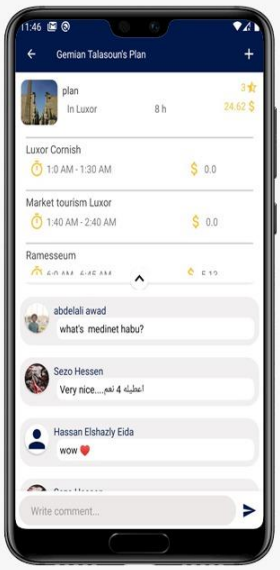


Figure 73: Dashboard Plan Screen

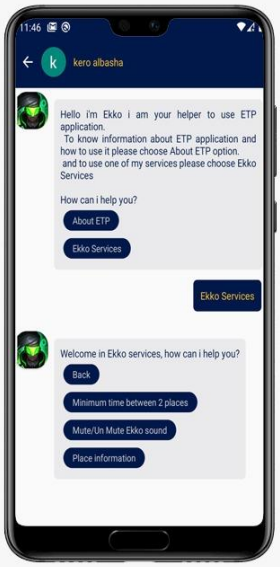


Figure 74: Ekko Screen

Figure 75: Place Info Screen

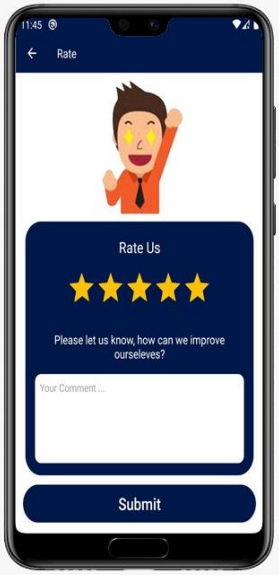


Figure 76: Rate App Screen

Figure 77: Contact Us Screen

Figure 78: About ETP Screen

3. PROTOTYPE DESCRIPTION

ETP is considered a mobile app used to create plan to user according to some constraints like preferences of him, budget and start and time to his travel, our system take this constraint and get information about places like opening closing, cost, type and so on …then app can output best fit plan to user correspond to his needs, can user modify on this plan by delete or add place to it. App has Ability that can user to share his plan to other users use app and this user have Ability to execute this plan and modify on it. ETP can show information about places that exit in the app. Finally, app have assistant called ekko can new user use it to show him how use app

3.1 IMPLEMENTATION PLATFORM

Technology	Implementation Platform
Mobile application	Android studio (Java/XML)
Online database	Firebase
Local database	JSON file
Authentication process	Google & Facebook API

Table 8 Implementation Platform

3.2 MAPPING BETWEEN REQUIREMENTS AND IMPLEMENTED FUNCTIONS

Functional requirements	**Functions/Class that implemented Functions requirements**
View user 's plan	Home class
View shared plans	Dashboard class
search about plan in dashboard	Dashboard class
Point of interests information	Placeinformationactitivty class
View plan	getplan()
Add place to plan	Additems()
Remove place from plan	Recyclerplanadapter class
Assistant services	Ekko class

Table 9 Mapping Between Requirements and Implemented Functions

3.3 IMPLEMENTATION DETAILS

ETP system contains main pages that provide several services based on their functionality, figure 13 describe these pages and their services.

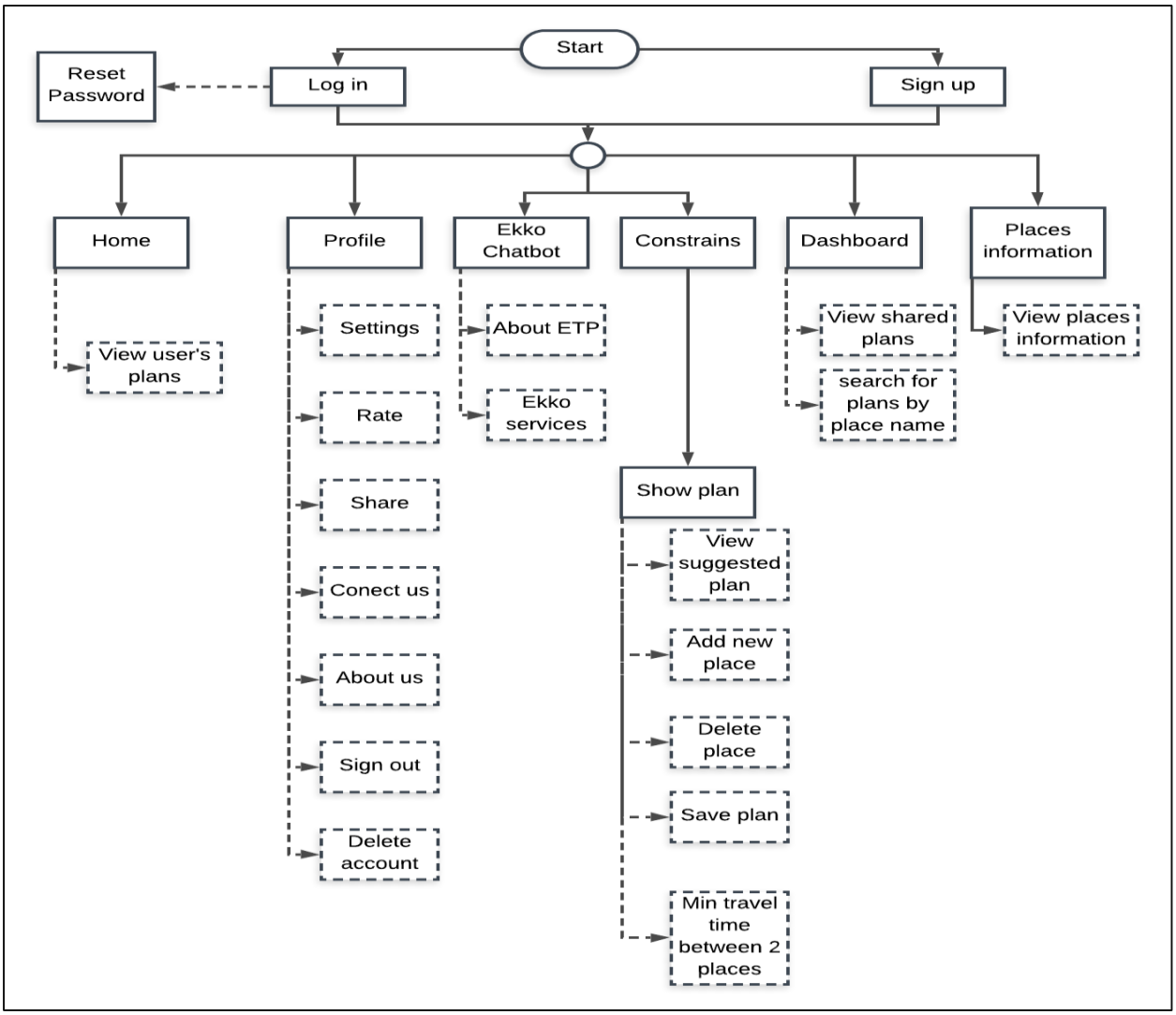


Figure 13 ETP Implementation Overview

In figure 13, the solid rectangle refers to the main page and the dotted rectangle refers to his services, the figure also describes the dependency between the pages, for example, the home page will not be viewed without login or sign up, the same thing in show plan page will not be viewed without constrains page and so on. Finally, the circle shape refer to the navigation bar which allowed the user to choose in which page he want to visit and changing between his children pages.
In the next sections, we will describe each of these pages with his services in detail based on figure 13.

Home Page:
The home page is the first page will be shown after logging in the system and his main functionality is listing user's plans, each user has a list of plans whatever these plans are old saved plans or imported from the dashboard, figure 65 shows home page user interface.

At first, the home page will get the current user's data (id, name, image, favorite list, forbidden list, preferences,...etc) from the login page and by using his id it will get all his plans whatever these plans are old saved plans or imported from the dashboard then the home page will view these plans in his layout. The home page will view each plan's information only (plan name, trip full cost, trip location, trip start time, trip end time) and by clicking on a specific plan it will view his places information (name, brief description, duration, minimum cost, start time, end time, image), also you can share or un-share a specific plan in the dashboard by using dashboard button.

Important functions in home:
- Share or un-share plan in dashboard

Sharing or un-sharing plan means publishing or removing it from the dashboard the main purpose for this service is that the plans generated by the algorithm may be used by more than one user and in future work may this service be developed to a recommendation system.
To share or un-share a specific plan, this plan has to belong this current user (he added and saved this plan). So that The system checks if the current user who added this plan, if he is he could use the service of share or un-share this plan otherwise he couldn't. Figure 14 shows share or un-share code.

```
//Share plan
holder.share.setOnClickListener((v) → {
        // Check if plan belong to current user
        if (mUploadedByList.get(position).equals(mCurrentUserData.getmCurrentUserId())) {...}
        else {
            Toast.makeText(context, text: "You cannot use this feature because you have not created this plan", Toast.LENGTH_SHORT).show();
        }

});
```

Figure 14 Upload trip to dashboard function 1

Only one copy of a specific plan could be shared on the dashboard. So that the user couldn't share the same plan multiple times in the dashboard.
Ekko Chatbot:

Ekko is a static chatbot with fixed questions and answers and his main functionality is improving ETP usability by acting as an information center or customer service center, and it shows that service by providing users with useful

information to use the application well and it provides a few services. Figure 74 shows the chatbot user interface Ekko exists in pages (Home, Dashboard, Places information and Constraints) in a fixed location and each of these pages has the same function that runs the chatbot. This function in the figure 15.
In the code of figure 15 the current page (Home,Constrains,…) call Ekko class which responsible for the chatbot running and how to interact with user, this part will be explained in the next section.

```
// *--------------------------------*  Call ekko chat bot    *--------------------------------*
public void ekko(){
    /* Part 1 */
    // Choose voice
    int x = new Random().nextInt( bound: 4);
    MediaPlayer ekkoSound;
    switch (x){...}
    ekkoSound.start();
    /* Part 2 */
    setContentView(R.layout.ekko_chat);
    LinearLayout layout = (LinearLayout) findViewById(R.id.chat_layout);
    ImageButton back = findViewById(R.id.ekko_back);
    /* Part 3 */
    // Call ekko class
    Ekko ekko = new Ekko( context: home.this,layout);
    /* Part 4 */
    // Set current user name and image
    CurrentUserData EkkoCurrentUserData = (CurrentUserData) getApplicationContext();
    final ImageView EkkoCurrentUserImage = findViewById(R.id.ekko_current_user_image);
    TextView EkkoCurrentUserName = findViewById(R.id.ekko_current_user_name);
    EkkoCurrentUserName.setText(EkkoCurrentUserData.getmUserName());
    if (mCurrentUserData.getmUserImage() != null)
    {...}
    //Return to pervious page
    back.setOnClickListener((v) → {
            startActivity(new Intent(getApplicationContext(), home.class));
            overridePendingTransition( enterAnim: 0,  exitAnim: 0);
            finish();
    });
}
```

Figure 15 Ekko chatbot function

Ekko Class:

Ekko class has the algorithm by which run the chatbot and interact with the user by fixed questions and answers, the class interacts with the user depending on a tree called Ekko tree, the main function of this tree is telling the class the current position and the next available positions. The tree is shown in figure 16.

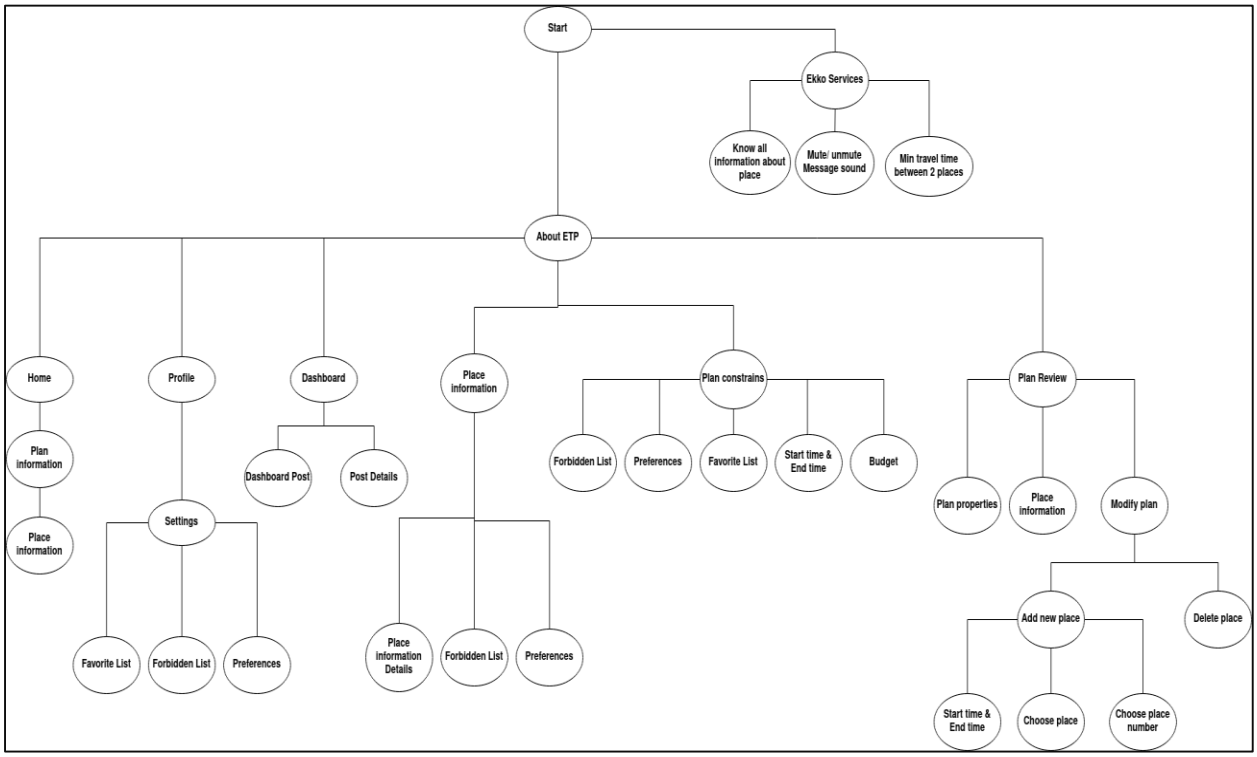


Figure 16 Ekko chatbot class overview

Ekko tree is an indirected tree used to detect what class has to do now and which options can be offered to the user next. The tree root is the start point in which the chatbot introduces himself and is considered as the current position then the chatbot view the next available positions as the next options which user could choose from, in our case will be (About ETP) and (Ekko Services). If the user wants to know information about the application and how to use it he has to choose (About ETP), and if he wants to use one of Ekko services choose (Ekko Services). For example, he chose (About ETP) option the chatbot will send a message “Here I can tell you important information about the app and how can you use it well. Which part in the application you want to know about?” and view the next options which in our case are (Home, Profile, Dashboard, Place information. Plan constraints and Plan Review) also it views (Back) option if the user wants to return

to the previous position in our case will be the start point. Then the user has to choose which page he wants to know about and if he chose (Home) option then the chatbot will view information about (Home) page and how the user can use it well. Again it will view the next options that will be (Back) and (Place information). (Place information) refers to any place information that viewed by clicking on a specific plan. This operation repeats itself as long as the chatbot running.

Each node in the Ekko tree represents a function in the Ekko class and when the user reached a specific node the class will call his function to answer the user question.

All node's functions have the same structure and the changes between each other in the values. Figure 17 shows the implementation code of the tree

```
// *--------------------------------*  Ekko class used to run the chat bot and provide all his services    *--------------------------------*
public class Ekko {
    /* Vars */
    private String message;         // Used to hold user or ekko message text
    Context context;
    LinearLayout layout;
    MediaPlayer sound;             // Used to hold message sound
    ArrayList<POI> POIs;           // Used to get all POIs in the system
    boolean ismute;                // Used to mute/unmute message sound
    // Start
    public Ekko(Context context,LinearLayout layout) {
        this.context = context;
        this.layout = layout;
        sound =  MediaPlayer.create(context,R.raw.message);
        POIs = POIsData(new ArrayList<Integer>(),new ArrayList<String>());
        ismute = false;
        start( i: 0);
    }
```

Figure 17 Ekko chatbot function 2

(message) is a global variable used by functions to represent the current text message, (POIs) is a list of all places in the system used by (Ekko Services) function to view places information, (ismute) is a boolean variable used to know if message sound muted or not, in the constructor after initializing variables, it starts with the root which is start node or start function.

As we said all functions have the same structure with differences in values. Figure 18 describes this structure in Ekko class

(Function-name) represents the name of the node tree that this function belongs to, (Function-id) used as an index for the function to be called with, the first part in function responsible for putting the current node text message in (message) variable,

```
/*
    // Function-name Function-id
    fun(){
        1- Ekko Message text
        2- List of next options
        3- make the view
    }
*/
```

Figure 18 Ekko chatbot explanation

second part make list of next available options based on the current node, last part sends all previous data to a function to make the message view and add it to the chat layout.

- Now we will represent an example function that used this structure in the figure 19

```
// modify 20
public void modify(){
    /* Part 1 */
    message= "You can modify output plan by:\n"+
            "- Delete place from plan\n"+
            "- Add new place in plan\n"+
            "Important note: If you will modify the output plan the application does not test the correctness of the plan " +
            "so that any changes on your own responsibility\n\n"+
            "Which modify process you want to know about?";

    /* Part 2 */
    ArrayList<String> options = new ArrayList<~>();
    //back
    options.add("Back-17");
    // add
    options.add("Adding new place in plan-22");
    //delete
    options.add("Deleting place from plan-21");

    /* Part 3 */
    reply(options,message, imgurl: 0);
}
```

Figure 19 Ekko chatbot function 3

We will take the (modify) function as an example, before the function declaration there is a comment that refer to (Function-name) in our case is modify which is a node in Ekko tree and (Function-id) which is 20 and it used as index for this function which allowed other functions to call this function, in (Part 1) the function sets his text message based on his current node, in our case the text message will be a description how the user can modify his plan, in (Part 2) it makes a list of next options based on Ekko tree and each item in list has 2 parts first part is a text describes the next option and second part is the id of the next option's function for example in "*options.add("Adding new place in*

plan-22");" code " *Adding new place in plan"* is the text that describe (add new place) option and *22* is the id of (add new place) function, in (Part 3) the function send the message and the list of options to (reply) function that make the message view and add it in the layout. The last parameter used if message will attach an image in the message or not in our case it won't. Figure 20 shows the (reply) function.

```
public void reply(final ArrayList<String> options,String message,int imgurl){
    /* Part 1 */
    // Get message layout
    View view = LayoutInflater.from(context).inflate(R.layout.ekko_message, root: null);
    LinearLayout messageLayout =(LinearLayout)view.findViewById(R.id.ekko_message_layout);
    TextView text = (TextView) messageLayout.findViewById(R.id.ekko_text);
    LinearLayout optionsLayout =(LinearLayout)messageLayout.findViewById(R.id.ekko_options);
    /*...*/
    if(imgurl==28){...}
    /* Part 2 */
    for(int i=0;i<options.size();i++) {
        // Divide list item string into the option text and option function id
        final String s = options.get(i).substring(0,options.get(i).indexOf("-"));
        options.set(i,options.get(i).substring(options.get(i).indexOf("-")+1));
        final int x,op;
        if(options.get(i).indexOf("-")!=-1) {...}
        else{...}
        final TextView button = EkkoButton();
        button.setText(s);
        button.setOnClickListener((v) → {
                if(!ismute)
                    sound.start();
                userMessage(s);
                button.setClickable(false);
                switch (x){...}
        });
        optionsLayout.addView(button);
    }
    text.setText(message);
    layout.addView(view);
}
```

Figure 20 reply function

In (Part 1) the function makes a new message view to change his values and add it to the group layout, in (Part 2) it loops in each next option and divides the string item of each option into the option text and option id. In the previous example, one of the options was "Adding new place in plan-22" so (s) variable will be “Adding new place in the plan” and (x) variable will be (22), (s) variable used to set the text that describes the option and (x) used to call the next option function when the user choose it by using his id, how this will happened? figure 21 shows the option listener. When the user chooses a specific option his listener will run that option’s function by using his id that stored in (x) variable. Finally, this message view will be added to the group layout.

```
final TextView button = EkkoButton();
button.setText(s);
button.setOnClickListener((v) → {
        if(!ismute)
            sound.start();
        userMessage(s);
        button.setClickable(false);
        switch (x){
            case 0:
                start(op);
                break;
            case 1:
                about();
                break;

            case 2:
                profilePage();
                break;

            case 3:
                settings();
                break;
```

Figure 21 Ekko chatbot Buttons

Constrains Page:

To add a new plan, the user has to send the plan constraints first then the algorithm uses these constraints to build the plan, Constraints page main functionality is to collect the plan constrains and send them to the plan review page that responsible for getting the plan. Figure 66 shows constrain page user interface.

The user has to send these constraints (plan name, trip start time, trip end time and trip full budget) and the rest of constraints are options. Also, the page tests if the constraints are valid or not such as (user has to fill required constraints, the start time has to be before the end time, budget is a double value).
(plan name) used to define the plan. (trip start time) and (trip end time) define the time range which the trip has to be in. (trip full budget) by this field the user sets the maximum cost he could pay during the trip, Note this cost does not contain travel cost between places but it only contains the Basic fees such as a ticket or entrance fees and so on. (Preferences) is a list of user's interests and hobbies the algorithm uses them to decide which places will satisfy user's personality to add

them in his plans. (Favorite List) is a list of places that the user interests to visit so that the algorithm makes them as high priority places and try to do his best effort to add them in his plans, note but that not means these places have to be in his plans because of some reason such as [this place not working on this time, traveling time between it and other places will exceed user's trip end time,...etc]. (Forbidden List) is a list of places that the user doesn't need to visit so that the algorithm deletes them from the user's available places to be visited so that it will not add them in his plans.

Plan Review Page:

After the user submits his plan's constrains the Plan Review Page will take this data to get his plan. Plan Review's main functions are generating the user's plan based on his constrains, allowing the user to modify the generated plan, providing the user with enough information to get a suitable plan and saving this plan in his list. Figures 71 shows the Plan Review user interface.

1- Get user's plan

The page will take plan's constrains that are sent by Constrains page and send them to Planner class to get user's plan, Planner class has the algorithm and it will be explained in the next section.

Figure 22 shows how Plan Review Page get the constrains and send them to Planner

```
//Get Plan
/* Get Constrains */
Time stTime = new Time(getIntent().getIntExtra( name: "startTimeHour", defaultValue: 0),getIntent().getIntExtra( name: "startTimeMin", defaultValue: 0)) ;
Time enTime = new Time(getIntent().getIntExtra( name: "endTimeHour", defaultValue: 0),getIntent().getIntExtra( name: "endTimeMin", defaultValue: 0)) ;
double bud = getIntent().getDoubleExtra( name: "budget", defaultValue: 0);
ArrayList<String> p = getIntent().getStringArrayListExtra( name: "prefrences");
ArrayList<Integer> favoriteList = new ArrayList<~>();
convertFromStringToInt(favoriteList,getIntent().getStringArrayListExtra( name: "favoriteList"));
ArrayList<Integer> forbiddenList = new ArrayList<~>();
convertFromStringToInt(forbiddenList,getIntent().getStringArrayListExtra( name: "forbiddenList"));
Data data = new Data(stTime,enTime,p,forbiddenList,favoriteList,bud);
/* Generate plan */
final long testEndTime,testStartTime;
testStartTime = System.currentTimeMillis();
plan = Planner.getPlan( MT: 10000, TabuFreq: 3, SFW: 1, TTW: 3, WTW: 2, MIWoutI: 10,data);
testEndTime = System.currentTimeMillis();
plan.getPlanD(POIStartTimes,POIEndTimes,POINames,POICosts,POIDurations,POIDescribtions,POIIDs,POIImageURLs);
```

Figure 22 Get user's plans function

At first Plan Review Page gets data that Constrains Page sent (Trip start time, Trip end time, Budget, Preferences, Favorite list, Forbidden list).

Important note: Each user has his own preferences, favorite list, and forbidden list and they stored in his record in database, also he can modify them from place information page or his profile and in each time he add a new plan, in constrains

page he will find preferences and favorite list and forbidden list are already selected based on which stored in database. Also, he can modify them in constrains page but these changes will not be saved in database. We consider these changes only for this plan, then the plan review page will take the last version of preferences, favorite list, and forbidden list which exist in constrains page.
The page will declare an instance of Data class with these data, this instance contains all data that the Planner class will need. Data class will be explained with Planner class. Finally, we calls (getPlan) function in Planner class with passing the instance of Data class to return user's plan which will be saved in (plan) variable and put plan's data in lists to be viewed in the page layout.

2- Modify plan
User can modify his generated plan by adding a new place in the plan or removing a place from the plan. To remove a place from the plan, the user has to make a long click on that place. And to add a new place in the plan he has to click on add button that exists in the top right corner of the page and the user has to fill the new place dialog, this dialog is shown in figure 71.

(Place start time) is the time when the user has to be in this place. (Place end time) is the time when the user has to leave this place. (Add place) the user has to choose the place he needs to add in the plan, note user here can choose the same place more than one time also he can add a place that exists in the forbidden list. (Add place number) the user has to choose where that new place be inserted in the plan.
Important note: any modification process in the generated plan is on user's responsibility only because the system doesn't make any validation tests on the modification process.
3- Save plan
To save this plan in user's list user has to click on save button but note user can't save an empty plan. Figure 23 shows an important part of save function.

```
// Save current plan
@Override
public void onClick(View v) {
    try {
        // Test if plan empty
        int test = 3/POINames.size();
        // Cal full cost
        double fullCost = 0;
        for (int i = 0; i < POICosts.size(); i++)
            fullCost += Double.parseDouble(POICosts.get(i));

        // Cal fullDuration, trip start and end times
        int fullDuration = 0;
        Time tripStartTime = new Time( hour: 23,  min: 59), tripEndTime = new Time( hour: 0,  min: 0);
        for (int i = 0; i < POIEndTimes.size(); i++) {
            int h, m;
            h = Integer.parseInt(POIStartTimes.get(i).substring(0, POIStartTimes.get(i).indexOf(":")));
            String part = POIStartTimes.get(i).substring(POIStartTimes.get(i).indexOf(" ") + 1);
            h = CF12T24(h, part);
            m = Integer.parseInt(POIStartTimes.get(i).substring(POIStartTimes.get(i).indexOf(":") + 1, POIStartTimes.get(i).length() - 3));
            Time startTime = new Time(h, m);
            h = Integer.parseInt(POIEndTimes.get(i).substring(0, POIEndTimes.get(i).indexOf(":")));
            part = POIEndTimes.get(i).substring(POIEndTimes.get(i).indexOf(" ") + 1);
            h = CF12T24(h, part);
            m = Integer.parseInt(POIEndTimes.get(i).substring(POIEndTimes.get(i).indexOf(":") + 1, POIEndTimes.get(i).length() - 3));
            Time endTime = new Time(h, m);
            fullDuration += Time.substract(startTime, endTime);
            tripStartTime = Time.min(tripStartTime, startTime);
            tripEndTime = Time.max(tripEndTime, endTime);
        }
```

Figure 23 Modify plan function

At first we test if plan is empty or not, (POINames) is a list of names for each place in plan and if the plan is empty so (test) variable will equal (3/0) “run time exception” so compiler leave the (try scope) and go to (catch scope) which view error message to user. Then, we calculate the trip full cost (POICosts) is a list of costs for each place in plan so that we get the sum of these values in (fullCost) variable. Also, we calculate trip full duration, trip start time and trip end time. We made Time class to interact with times. We initialize (tripStartTime) with the maximum time possible on the other hand (tripEndTime) with the minimum time possible. (POIStartTimes) is a list of start times for each place in plan and (POIEndTimes) is a list of end times for each place in plan. then we loop for each item in (POIEndTimes). (CF12T24) is a function to change time format from 12 to 24 and for each place (startTime) variable store the start time for that place and (endTime) store the end time, (subtract) function in Time class return the subtract of end time and start time in minutes and if start time bigger so it return -1. Full duration refer to the full time user spends in places so that (fullDuration) variable is the sum of the subtract between end time and start time for each place. (min) function in Time class return the minimum time of 2 parameters times and (max) return the maximum. So that (tripStartTime) will store the minimum start time of places and (tripEndTime) will store the maximum end time of places. In the rest of the code we will save all these information in database.

Planner Class:
This class has the algorithm which makes the user's plan based on his constrains. As seen before the function used to get the plan is called (getPlan) and this function exists in Planner class. Planner class or (getPlan) function divided into 3 main parts. Figure 24 shows these parts with his main functions and the information which are sent between these parts.

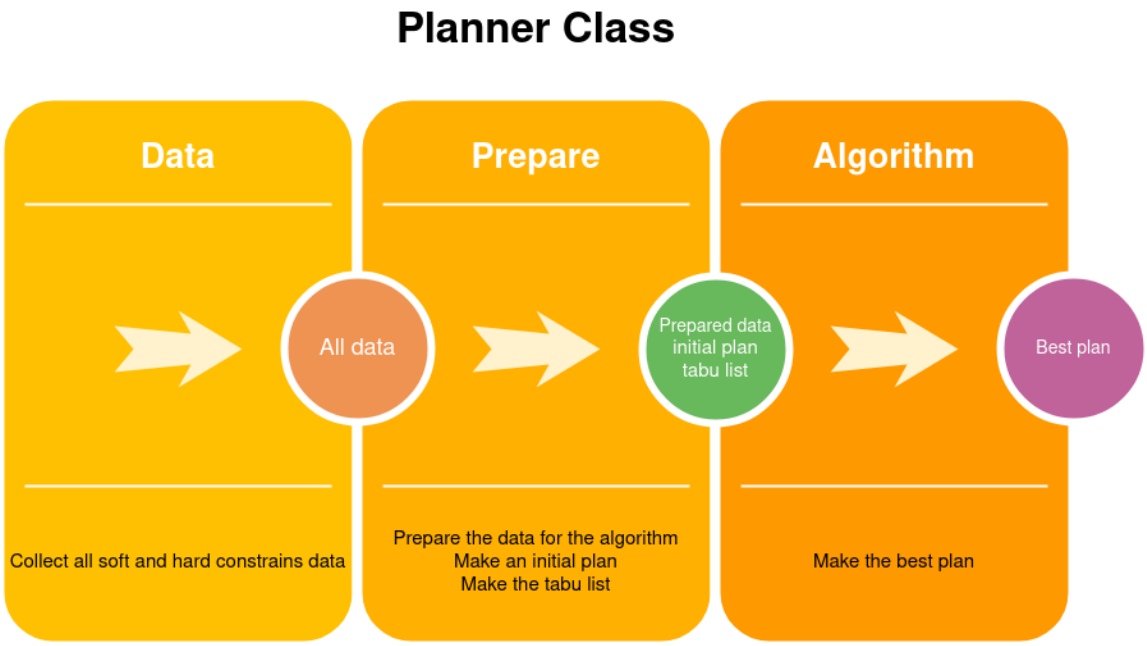


Figure 24 Planner class

Now we will explain all these parts in detail in the next sections.

1- Data At first, we need to get all the data that the algorithm will need and this is the (Data class) main function. Actually we divide these data into 3 main parts (Point Of Interest "POI" Data), (Trip Data) and (User Data). Figure 25 shows these parts and which specific data exists into each part.

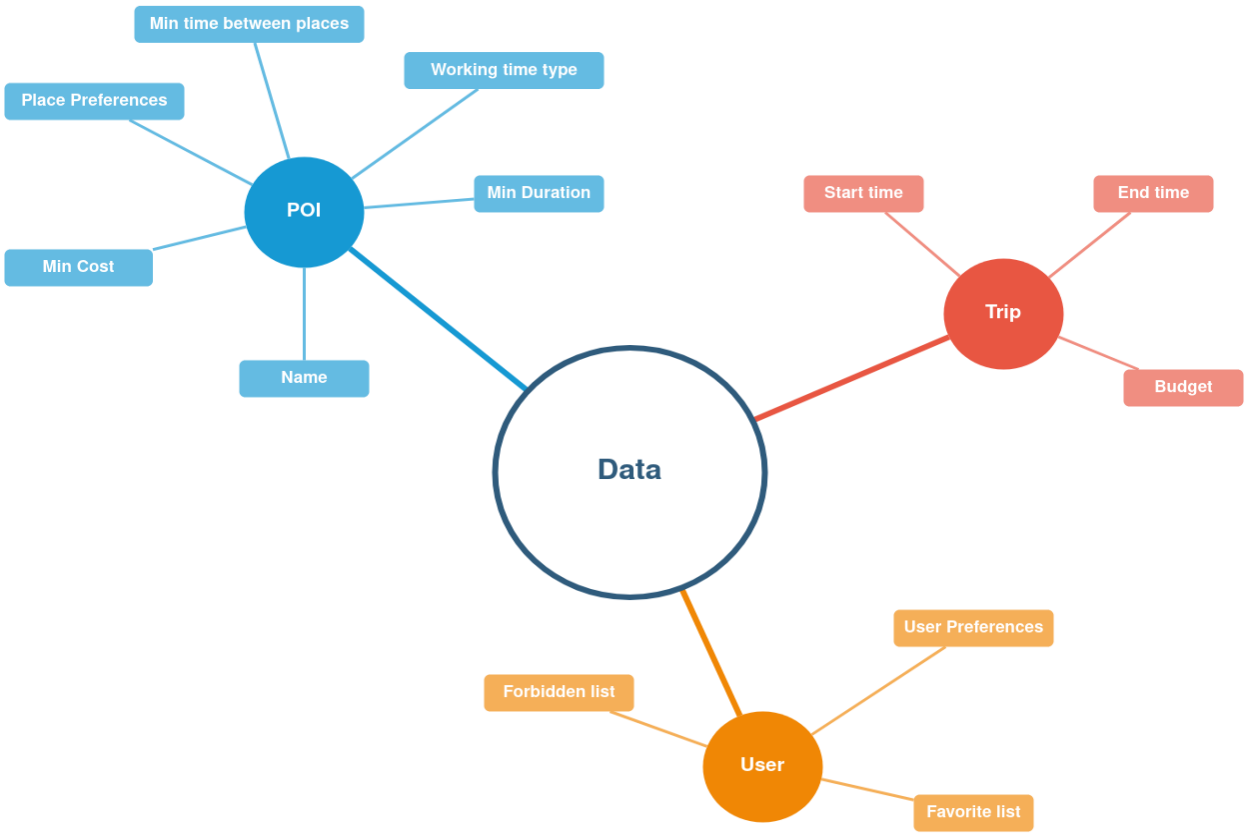


Figure 25 data

In (POI part) we get all places data that is stored locally in the JSON file and for each place we get, his name, minimum cost "Basic fees", place preferences (it is a list of which things can the user do in that place like walking, running or swimming), minimum travel time between that place and all other places (this stored manually based on google maps), working time type (each place in the system divided into a specific part based on his working time and all parts are (M) represents the morning part, (N) represents the night part and (A) represents full part places where work in both time for example, if a place works from "06:00 AM" to "11:00 AM" it in (M) part), the minimum duration (the minimum time that the user can spend in that place and this time is just a suggestion from us).

In (Trip part) we get the trip data which filled in the constraints page such as start time, end time, and the maximum budget.

In (User part) we get the data that related to the user. We get them from constrains page such as user preferences, which it is a list of user's hobbies and interests. The favorite list, which is a list of places user wants to visit. And the forbidden list, which is a list of places user won't visit.

Data class-main function is to collect all these data to be used by the next parts (prepare and algorithm). The Data class code is shown in figure 26.

```
// *--------------------------------*  Data class used to get all data used in planner algorithm (User-Trip-POIs)    *--------------------------------*
public class Data {
    /* Vars */
    public Time startTime;
    public Time endTime;
    public ArrayList<String>userPrefrences;
    public ArrayList<Integer>userForbiddenPOIs;
    public ArrayList<Integer>userFavoritePOIs;
    public  double budget=0;
    // *--------------------------------*  Get user and trip Data from constrains page    *--------------------------------*
    public Data(Time startTime, Time endTime, ArrayList<String> userPrefrences, ArrayList<Integer> userForbiddenPOIs,
                ArrayList<Integer> userFavoritePOIs, double budget) {
        this.startTime = startTime;
        this.endTime = endTime;
        this.userPrefrences = userPrefrences;
        this.userForbiddenPOIs = userForbiddenPOIs;
        this.userFavoritePOIs = userFavoritePOIs;
        this.budget = budget;
    }
    // *--------------------------------*  Return user data (preferences, favorites, forbidden)   *--------------------------------*
    public User userData(){...}
    // *--------------------------------*  Return trip data (start time, end time , budget)    *--------------------------------*
    public Trip tripData() { return ( new Trip(startTime,endTime,budget, startPOI: null, endPOI: null) ); }
    // *--------------------------------*  Return POIs from json file     *--------------------------------*
    public static ArrayList<POI> POIsData(ArrayList<Integer>favorite,ArrayList<String>userPreferences){...}

}
```

Figure 26 data class and function

As we mentioned before Data class get (Trip) and (User) data from constrains page using his constructor and return them with (userData) and (tripData) functions. (POIsData) function it return a list of (POI) where each place in the system represented in POI object. Figure 27 represents the information stored for each POI object.

```
// *--------------------------------*  Used to present Point of interest (Place)   *--------------------------------*
public class POI {
    private int id;                                 // POI ID
    private String name;                            // POI Name
    private double cost;                            // POI Minimum cost
    private ArrayList<String> preferences;          // POI Preferences
    private Map<Integer,Integer> shortestPathes;    // POI Shortest Pathes to other POIs
    private String type;                            // POI Working time type (M,A,N)
    private Integer duration;                       // POI Minimum spend time
    private boolean isFavorite;                     // If this POI in user's favorite list
    private double value;                           // POI satisfaction factor
    private Time closeTime;                         // POI Close Time
    private Time openTime;                          // POI Open Time
    public String desc;                             // POI Desciption
    public String photoURL;                         // POI photo URL
```

Figure 27 Pois Variables

We will explain only variables which didn't explained before because most of these variables already explained in figure 25**.** (id) is the place identifier in the system. (shortestPathes) is a map that stores for each place the place id and the minimum travel time to that place. (isFavorite) is a boolean variable his value is true when that place exists in the favorite list that sent from the constrains page otherwise false.

(value) is an evaluation value for this place and we get this value by matching the user preferences and the place preferences also if that place in the favorite list we increase his value for example, user preferences are
"walking", "photography", "swimming" and place preferences are
["photography","Grilling"] so that place value will be 1 and if this place in favorite list we increase it by 10 to be 11. (closeTime) and (openTime) are the working time for that place. (desc) is a brief description for that place.

Plan constraints divided into 2 main parts soft constraints and hard constraints we use the hard constrains to decide if the plan is a candidate or not and by using the soft constraints we evaluate each candidate plan to choose the best candidate plan. Figure 79 shows that division.

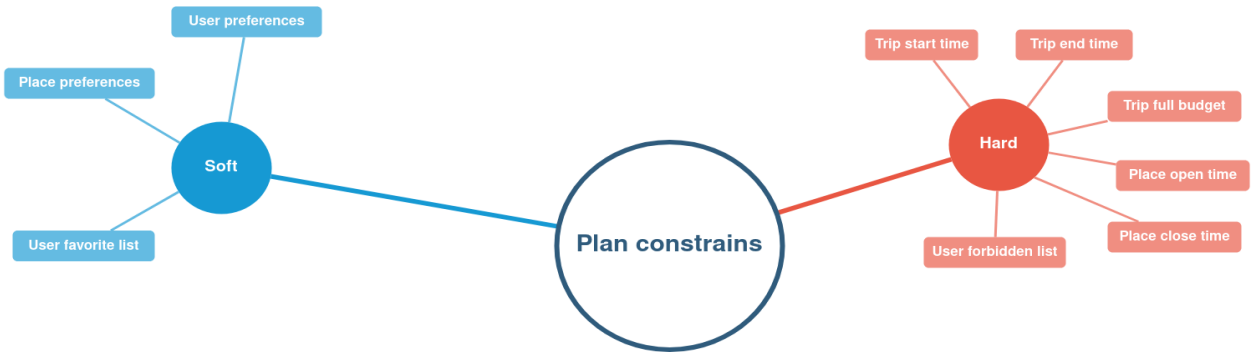


Figure 79 How To Choose the best candidate plan

2- Prepare

After getting all the important data we will send them to the next phase which is Prepare phase. In this phase, we do some important preparing processes before starting the algorithm and we do them using a class called (Prepare class). (Prepare class) main functions are preparing the data, make an initial plan, and make the tabu list. In next sections we will explain each function in detail.

A. Prepare the data
Figure 80 views the steps of this phase.

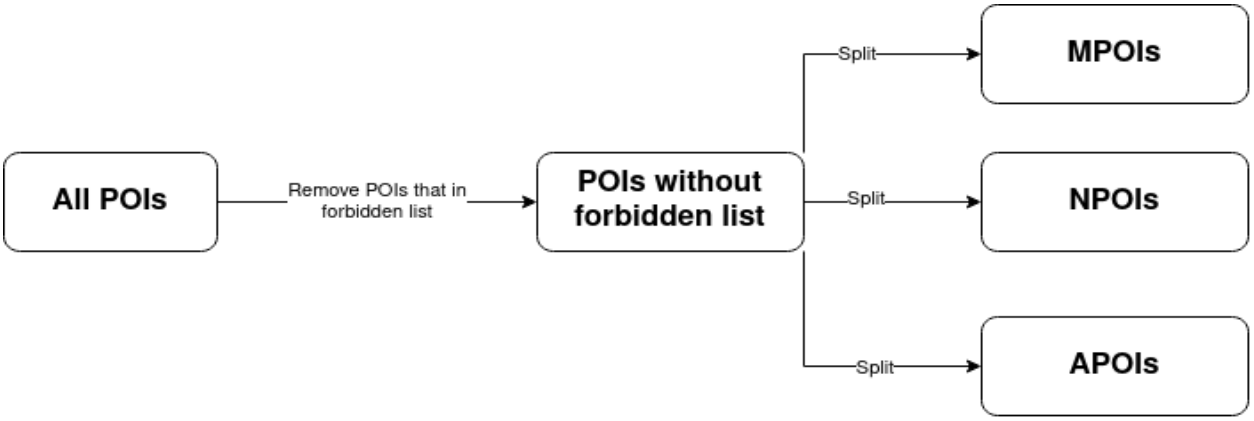


Figure 80 View the steps of this phase

After getting all places in the system, the prepare class has to remove places that exist in the user's forbidden list. Instead of working on one list has all left places we will divide this list into 3 lists depending on working time type. This phase implementation code in the figure 28.

```
public Prepare(ArrayList<POI> POIs,Trip trip,User user){
    // When morning part ends
    MTimeEnd= new Time( hour: 15, min: 0);
    // When night part ends
    NTimeEnd= new Time( hour: 23, min: 59);
    MPOIs=new ArrayList<POI>();
    NPOIs=new ArrayList<POI>();
    APOIs=new ArrayList<POI>();

    // Remove Forbidden list from out POIs
    for(int index:user.getForbidden()){
        POIs.remove(POIs.get(index-1));
    }

    // Divide POIs depending on type
    for(POI poi:POIs){
        if(poi.getType().equalsIgnoreCase( anotherString: "M"))
            MPOIs.add(poi);
        else if(poi.getType().equalsIgnoreCase( anotherString: "N"))
            NPOIs.add(poi);
        else
            APOIs.add(poi);
    }
```

Figure 28 prepare the date

In Prepare class constructor, we put when morning and night parts end in code above based on 24-time format, morning part ends at “15:00” and night part ends at “23:59” and that means any place is opened and closed in the time range from “00:00” to “15:00” is morning working time type (M) and any place is opened and closed in the time range from “15:01” to “23:59” is night working time type (N) otherwise is full working time type (A). (POIs) which is the list of all places in the system. Then we loop for each item in the forbidden list and remove that place from (POIs) using the place’s id. Then we divide the remaining places in (POIs) into (MPOIs) “Morning Point Of Interests”, (NPOIs) “Night Point Of Interests” and (APOIs).

The next function in Prepare class is making the initial plan but before explaining this function we have to explain how we represent the plan in the system.

In the algorithm, we divide the plan into 2 main parts as sub plans and these parts are Morning sub-plan and Night sub-plan. The plan which contains the 2 sub-plans (morning and night) represented by a class called (FullPlan) and any sub-plan represented by a class called (Plan). (FullPlan) and (Plan) are Doubly linked lists so that each place in the plan represented by a node with 2 references and data which is the place. Figure 81 shows how the system represents the plan and which places could be inserted into each sub plans.

Figure 81 shows the container (FullPlan) class that represents the plan in the system has 2 sub plans (Morning) and (Night) sub plans which represented by (Plan) class. We can only insert MPOIs “Morning places” and APOIs “Full-time places” in the morning plan and we can only insert NPOIs “Night places” and APOIs in the night plan.

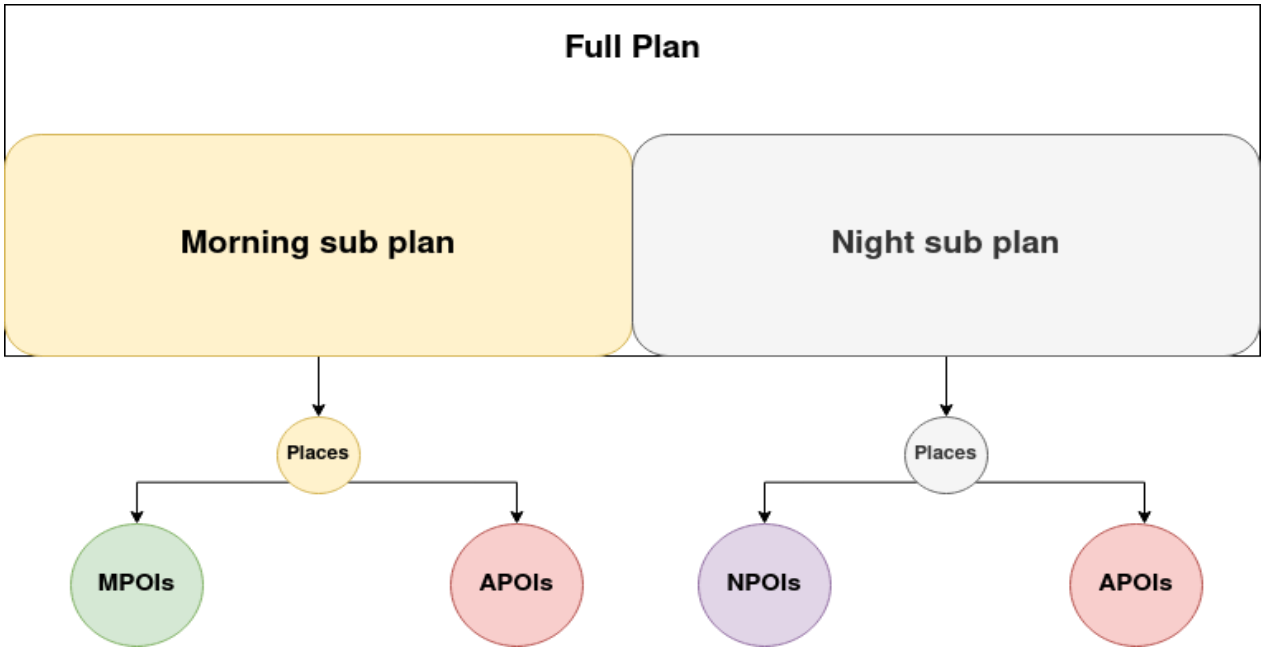


Figure 81 Night and Morning POIs

Figure 30 shows the plan class and its variables.

```
// *--------------------------------*  Used to present sub plan   *--------------------------------*
public class Plan {
    private Node trip;          // start pointer node
    private int NOV;            // number Of Visites
    private Integer fullTime;   // plan full time in minutes
    private double fullCost;    // plan full cost
    private Integer travelTime; // plan full travel time in minutes
    private int wasteTime;      // plan full waste time in minutes
    private double SFs;         // plan full satisfaction factor


    public Plan(Time startTime,Time endTime) {...}
```

Figure 30 Plan class and its variables

Plan class is a Doubly linked list and his places represented by nodes. In the next section we will explain the node class. The class contains this information, the start pointer node which points to the first place in the plan. the number of places will be visited in the sub-plan. the full time and the full cost for this sub-plan. the full travel time for this sub-plan which is the time user needed to travel between places. the full waste time for this sub-plan and this time contain 3 scenarios, when the user arrived at a specific place before it is opened, when the trip finished but there is time left and the subtract time between the trip start time and the first place he will visit in the plan. the satisfaction factor for this sub-plan which is the sum of sub-plan places values.

Figure 31 shows the class node.

```
// *--------------------------------*  Used to present POI in plan   *--------------------------------*
public class Node {
    public Node before;         // point to pervious node
    public Node next;           // point to next node
    public POI poi;             // this node POI
    public Time from;           // start time in this place
    public Time to;             // end time in this place

    public Node() {
    }

    public Node(Node before, Node next, POI poi,Time from,Time to) {
        this.before = before;
        this.next = next;
        this.poi = poi;
        this.from=from;
        this.to=to;
    }

}
```

Figure 31 node variables and constrictor

This node class represents the place in the plan. Because (FullPlan) and (Plan) are Doubly linked lists the node has 2 pointers the next and the before. It has the place (POI) as his data or value. Also, it has the start time which is the time the user has to be in that place based on the plan the same in the end time but instead leaving that place.

```
// *--------------------------------*  FullPlan class used to represent the full plan that contains Mplan and Nplan subplans   *--------------------------------*
public class FullPlan {
    public Plan MPlan;          // Morning sub plan
    public Plan NPlan;          // Night sub plan
    public PlanStructure planData;  //full plan data

    public FullPlan(Plan MPlan, Plan NPlan) {
        this.MPlan = MPlan;
        this.NPlan = NPlan;
        this.planData = new PlanStructure();
    }
```

Figure 32 full plan class (1)

Figure 32 shows the FullPlan class which represents the plan in the system and it contains 2 main sub-plans the morning sub-plan (MPlan) and night sub-plan (Nplan). And it has an instance of the (PlanStructure) which is a data structure used to save plan's important information and these information shown in Figure 33.

```
// *--------------------------------*  Used to present full plan data   *--------------------------------*
public class PlanStructure {
    public int wasteTime;       // plan full waste time
    public int travelTime;      // plan full travel time
    public double fullCost;     // plan full cost
    public double SFs;          // plan full satisfaction factors
    public int NOVs;            // plan number of visits
}
```

Figure 33 full plan class (2)

B. Make initial morning and night sub-plans is shown in figure 34.

```
/* Make Morning Intial plan */
MinitialPlan=new Plan(trip.getStartTime(),trip.getEndTime());
Time currentTime = new Time(trip.getStartTime().hour,trip.getStartTime().min);
makeSubPlan(MinitialPlan, currentTime, MPOIs, MTimeEnd, trip, mPlan: null, skip: false);

// Check if current time passed morning end time
if(currentTime.compare(MTimeEnd)){
    makeSubPlan(MinitialPlan, currentTime, APOIs, MTimeEnd, trip, mPlan: null, skip: false);
    if(currentTime.compare(MTimeEnd))
        makeSubPlan(MinitialPlan, currentTime, MPOIs, MTimeEnd, trip, mPlan: null, skip: true);
}


/* Make Night plan */
NinitialPlan=new Plan(trip.getStartTime(),trip.getEndTime());
makeSubPlan(NinitialPlan, currentTime, NPOIs, NTimeEnd, trip,MinitialPlan, skip: false);

// Check if current time passed night end time
if(currentTime.compare(NTimeEnd)){
    makeSubPlan(NinitialPlan, currentTime, APOIs, NTimeEnd, trip,MinitialPlan, skip: false);

    if(currentTime.compare(NTimeEnd))
        makeSubPlan(NinitialPlan, currentTime, NPOIs, NTimeEnd, trip,MinitialPlan, skip: true);
}
```

Figure 34 Initial plan

At first, we make the morning initial plan by initializing an empty plan (MinitialPlan) and a Time object to detect the current time and it initialized by trip start time. Then we call (makeSubPlan) function to try inserting places in it.
(makeSubPlan) function takes these parameters that shown in figure 82, the sub-plan. the current time and inside the function, if it inserted a new place in the plan it also updates the current time. The list of places which the function will try inserted places from. Because we working with morning and night sub-plans we have to send when they end. The user's trip constraints (start time, end time, budget). (mPlan) this parameter used to connect these 2 sub-plans for example, if the morning plan has places and we need to check if we can insert a place in the night sub-plan or not. In cost check this condition has to be true
(place.minimumCost + NinitialPlan.fullCost + MinitialPlan.fullCost <= trip.fullBudget)
so that we need the (MinitialPlan) information in making the night plan. The (skip) parameter is a flag used to detect if the function allows a waste time or not in inserting the places.

```
// *--------------------------------*  Return legal sub plan  *--------------------------------*
private void makeSubPlan(Plan plan,Time currentTime,ArrayList<POI> POIs,Time timeEnd,Trip trip,Plan mPlan,boolean skip){...}
```

Figure 82 Make Sub Plan

At first (makeSubPlan) call, we send MPOIs and while the current time didn't be bigger than the morning time ends, we send APOIs then MPOIs again.

Then we make the night initial plan with the same style.
In the figure 35 we show the algorithm used in (makeSubPlan) function

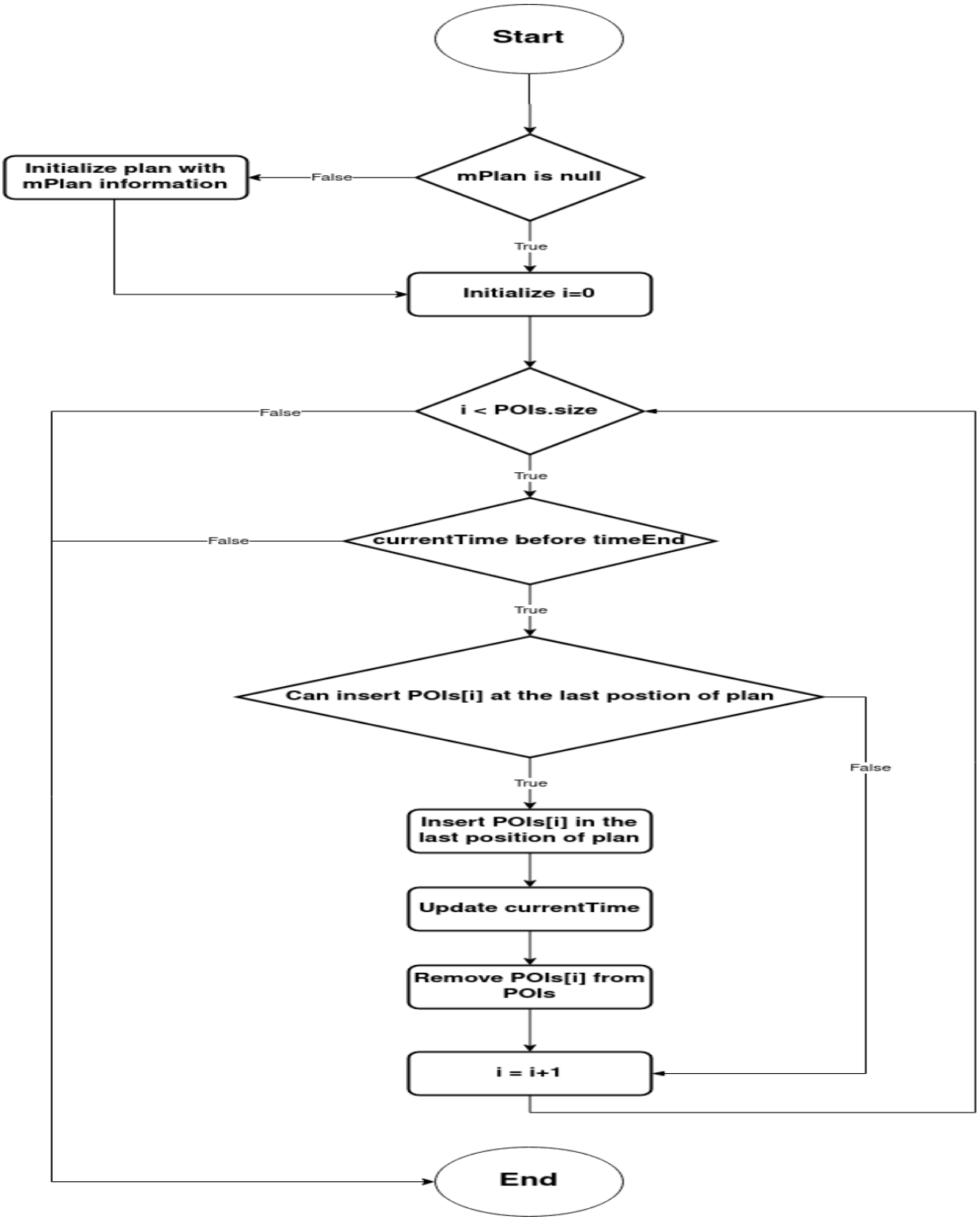


Figure 35 flowchart greedy algorithm

We are using a greedy algorithm to make the initial plan that his flow chart shown in figure 35.
Depending on the parameters we already explained before and the signature in figure 82 if (mPlan) in not null we have to connect this night plan to the morning plan. Then we loop in each place of (POIs) list and checking in each iteration if the current time passed the end time based on morning or night, if yes we end the function. Then we check if we can insert that place in the last position of plan or not and this check made by another function called (canInsertLast) this function return boolean value and it return true if place cost and time addition to plan full cost and time satisfy these constrains:
1- place cost + plan full cost <= budget
2- plan full time + place min time <= trip end time
3- plan full time + place min time <= place close time
4- if skip is false means the waste time not allowed so
currentTime >= place open time
otherwise the function return false.
And if function return true then we insert that place at the last position of plan and update the current time and remove that place from (POIs).

C. Make the tabu list
After making the initial morning sub-plan and the initial night sub-plan by the prepare class. Now we have the initial full plan and we have to send this initial plan to the algorithm to get the best plan. The algorithm we used to get the best plan is the tabu search algorithm. In this algorithm, we have to use a structure called the tabu list.
In the tabu search, we get the neighbors of the current state then we choose the best, but to avoid the repetition which may cause an infinity loop we use the tabu list.
The tabu list has 2 main functions in our algorithm are avoiding the repetition of neighbors which may cause the infinity loop and avoiding the local maximum problem. So we have to make the tabu list before starting the algorithm.
To get the neighbors of the current plan we use 3 operations swap, insert and delete. We have to store the operation made on the plan in the tabu list for a specific iterations then delete this operation from the tabu list. Figure 36 shows the Tabu list class.

```
// *--------------------------------*  Tabu list structure  *--------------------------------*
public class TabuList {
    // id1-> poi id || id2 -> freq || poi
    private ArrayList<TabuPair> Dlist;          // Delete List
    // id1 -> poi id || id2 -> index || poi=null
    private ArrayList<TabuPair> Ilist;          // Insert List
    private int[][] Igraph;                     // Insert graph
    // id1 -> poi1 id || id2 -> poi2 id || poi=null
    private ArrayList<TabuPair> Slist;          // Swap List
    private int[][] Sgraph;                     // Swap graph

    public TabuList (){
        //maximum number of POIs
        int MNOPOIs = 101;
        //maximum number of visits
        int MNOVs=101;
        Dlist = new ArrayList<TabuPair>();
        Ilist = new ArrayList<TabuPair>();
        Igraph = new int[MNOPOIs][MNOVs];
        Slist = new ArrayList<TabuPair>();
        Sgraph = new int[MNOPOIs][MNOPOIs];
    }
```

Figure 36 Tabu class

To store delete operations in the tabu list we use a list of objects from class (TabuPair). (TabuPair) is a data structure class (like a struct in c++) has 3 attributes 2 integer variables and an object of POI and this class used only in tabu list class. For easy explaining let call the first integer (id1) and the second (id2) and the object as poi.

And to store insert and swap operations we used a list of (TabuPair) and 2-dimensional array for each.

In the constructor, we initialize their values. (MNOPOIs) “Maximum Number Of Point Of Interests” defines the maximum number of POIs that could be stored in the system which in this case are 100 because the POI id starts from 1. And (MNOVs) “Maximum number of visits” defines the maximum number of places in any plan which in this case 100 too. Note these values could be changed as long as they don’t bigger than the maximum size of the 2-dimensional array. The figure 37 shows the tabu list structure with an example.

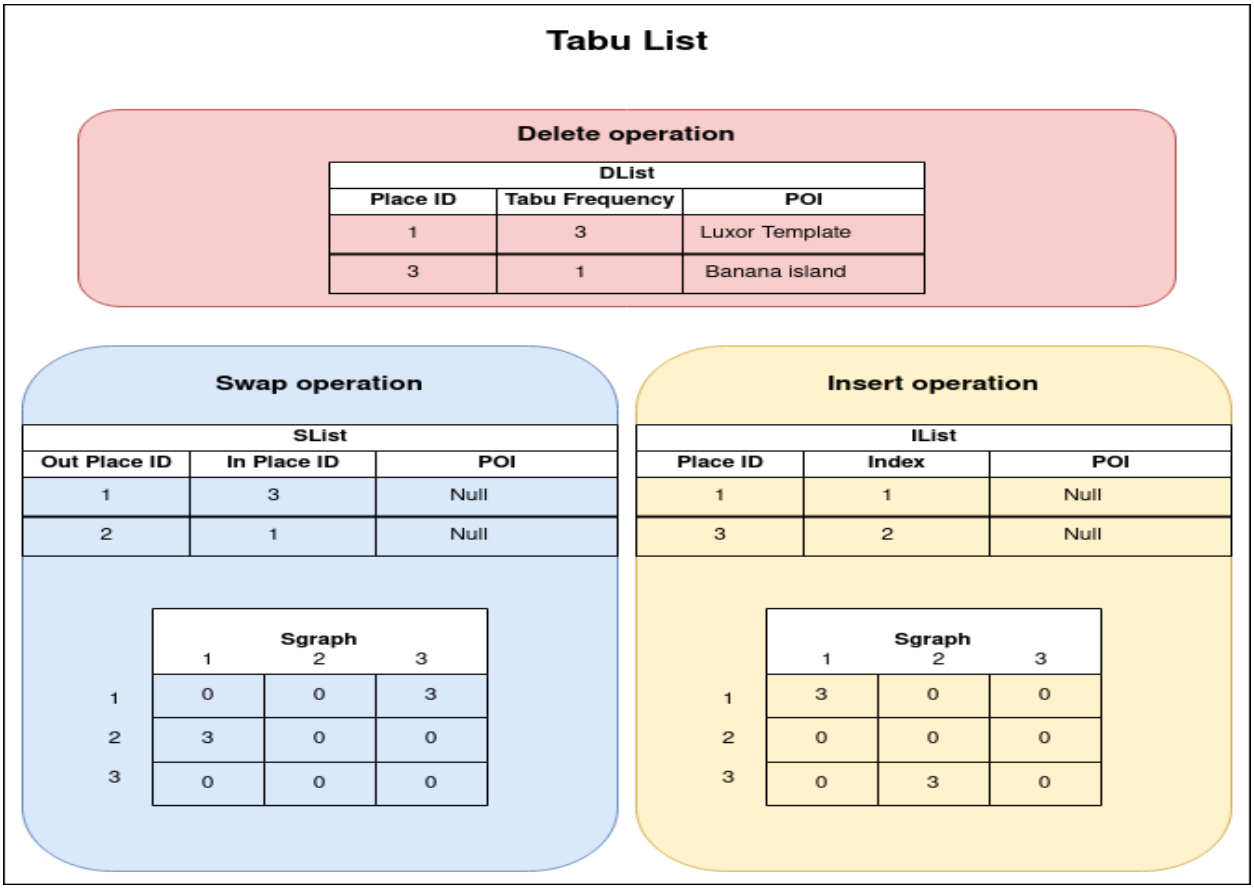


Figure 37 tabu list

To store a delete operation in the tabu we use a list (TabuPair). We use (id1) to represent the place id. (id2) to represent the frequency number. The poi to store the object of the POI because when we delete a POI from the plan that already deleted from the original POIs list, we could return this POI when his frequency number be 0. let the frequency number is 3, in the example we made 2 delete operations, Luxor template with id 1 and the banana island with id 3 each of them starts his frequency number value with 3. in each iteration, we update the tabu list and decrease their values by 1. When frequency value be 0, we return this POI in POIs list and remove this operation from the tabu list. And because Luxor template frequency is 3 and the island is 1 we conclude that the island removed before the template by 2 iterations. The code in figure 83 shows how tabu insert new delete operation.

```
public void Dinsert (POI poi,int freq){
    Dlist.add(new TabuPair(poi.getId(),freq,poi));
}
```

Figure 83 How tabu insert new delete operation

To store a swap operation we use a list (TabuPair) and a 2-dimensional array. In the swap operation, we swap each place that not in the plan with a specific place in the plan. In the list, we use (id1) to represent the id of place does not exist in the plan, (id2) to represent the id of place exists in the plan and the poi will be null. To insert a new swap operation we insert their id in the list and change their value in the graph with the frequency value. In the first operation of the example, we made a swap operation between place not in plan with id 1 and place in plan his id 3 so we inserted their values in the list then we changed the value of graph[1][3] from 0 to frequency value which in our case is 3.We use this graph to check if the operation exists in the tabu list or not if his value in the graph is 0 that means not exist in the tabu list otherwise it exists. Like delete operation we update the tabu list at each iteration by decreasing values bigger than 0 by 1. And when their values are 0 in the graph we remove this operation from the list. We use the list in update processes instead of checking for each item in the graph to decrease by the list we go directly to the items that have values bigger than 0 in the graph. Figures 84 shows, insert a new swap operation and search if this swap operation in the tabu or not.

```
public int Ssearch(int id1,int id2){
    if(id1>id2){
        int temp=id1;
        id1=id2;
        id2=temp;
    }
    return Sgraph[id1][id2];
}
public void Sinsert (int id1,int id2,int freq){
    if(id1>id2){
        int temp=id1;
        id1=id2;
        id2=temp;
    }
    Sgraph[id1][id2] = freq;
    Slist.add(new TabuPair(id1,id2, poi: null));
}
```

Figure 84 Insert a new swap operation

The swap and the insert operations have the same style but the difference between them is the 2 integer variables. In the insert, (id1) represents the place id and (id2) represents the index which that place inserted in the plan.

3- Algorithm

In (getPlan) function it has the algorithm used the get the best plan from the initial and this function combines the 3 main parts (Data, Prepare and Algorithm). This function implements figure 24 and the code in figure 38 shows the first 2 phases.

```
// *--------------------------------*  Get plan algorithm   *--------------------------------*
public static FullPlan getPlan(int MT,int TabuFreq,double SFW,double TTW,double WTW,int MIWoutI,Data data){

    /*Get Data (User, Trip, POIs)*/
    User user=data.userData();
    Trip trip=data.tripData();
    ArrayList<POI> POIs = Data.POIsData(user.getFavorite(), user.getPreferences());

    /*Prepare Data*/
    //1- Remove forbidden list
    //2- Split POIs depending on type
    //3- Make initial MPlan and NPlan
    //4- Make Tabulist
    Prepare pre = new Prepare(POIs, trip, user);
    TabuList tabuList = new TabuList();


    /*Algorithm*/
```

Figure 38 get plan function

At first, we implement the first phase which responsible for getting the data. Then we implement the second phase which responsible for preparing. And the last step we implement the algorithm. Please remember function's parameters because we will use them in the algorithm except for (data) parameter which used in getting the data phase.

Each plan be evaluated by a function called (evaluatePlan) and this function checks if the plan valid and satisfy trip constrains, it returns plan's value otherwise returns -1.

The function use this equation to get plan's value:

SFW * SFs + TTW * TT +WTW * WT

where:

SFW "Satisfaction Factor Weight" it is a double value is sent as parameter in the function and used to set the satisfaction factor priority

SFs "Satisfaction Factors**"** it is the sum of each place satisfaction factor in the plan

TTW "Travel Time Weight**"** it is a double value is sent as parameter in the function and used to set the travel time priority

TT "Travel time**"** it is the complement of the plan's full travel time which his equation is :

TT = 1 – (full travel time / trip full time)

WTW "Waste Time Weight**"** it is a double value is sent as parameter in the function and used to set the waste time priority

WT "Waste Time**"** it is the complement of the plan's full waste time which his equation is : **WT** = 1 – (full waste time / trip full time)

Figure 39 shows the flow chart of the algorithm used to generate the plan using the tabu search.

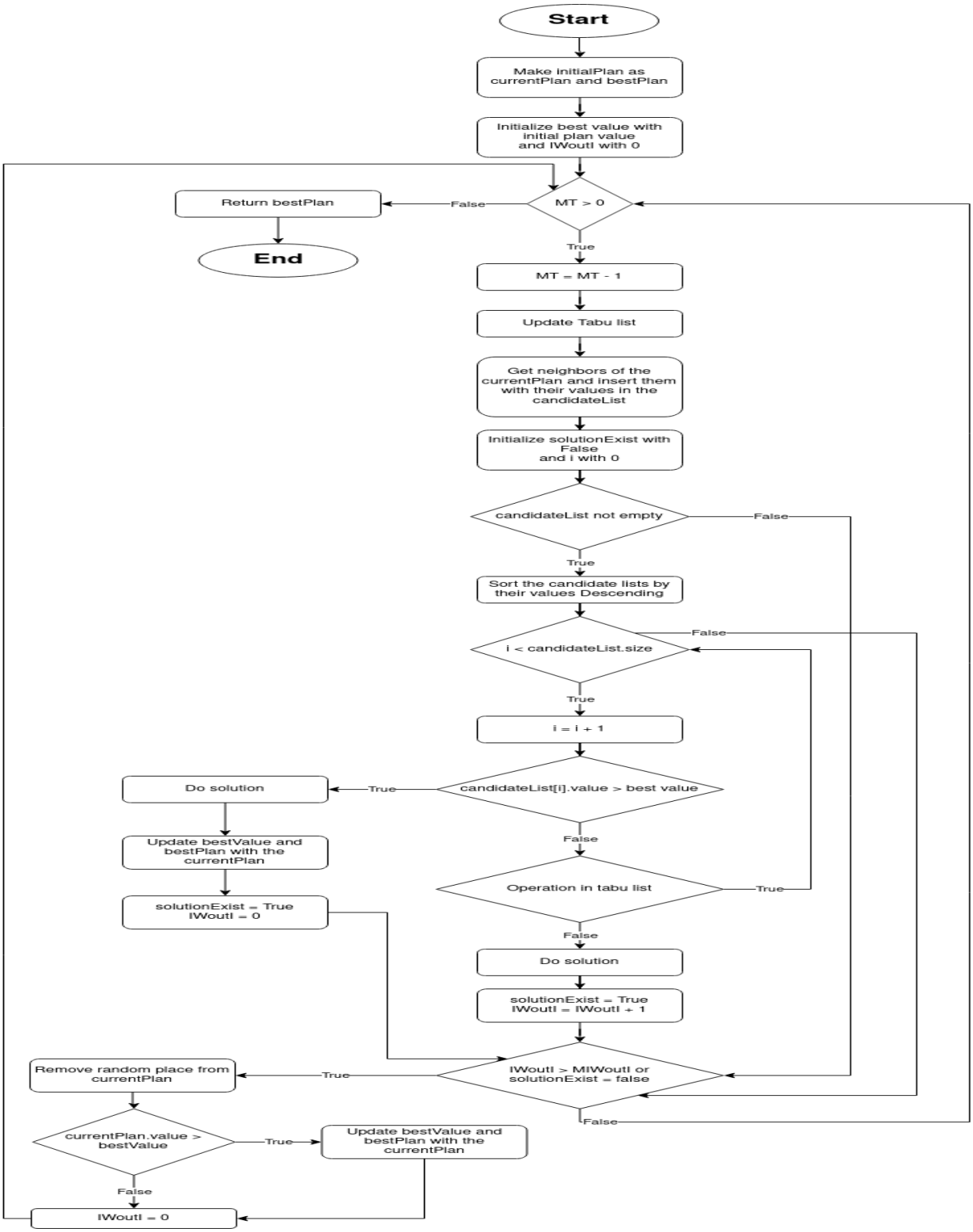


Figure 39 flowchart of make plan algorithm

(bestPlan) is a plan used to save the best plan during the algorithm. And the (bestValue) variable save (bestPlan) evaluation value. Also we use an integer variable called (IWoutI) “Iterations Without Improvement” this variable used to count the number of times the plan changed but didn’t make any Improvement --- bestValue didn’t change--- and if this counter be bigger than the variable (MIWoutI) “Maximum Iterations Without Improvement” we delete a random place from the currentPlan. At first we initialize the (bestPlan) and the (currentPlan) with the the (initialPlan) that made by Prepare class also we initialize bestValue with (initialPlan) value. (MT) “Maximum tries” is an integer variable is sent as a function’s parameter used to decide the number of iterations we repeat the algorithm. So we do the algorithm (MT) times and in each time, we update the tabu list. Get the neighbors of the (currentPlan) and each neighbor is a different plan then we save these neighbors and their plan’s values in a list called (candidateList). (candidateList) is a list of objects from class called (CandidatePair) which is a class like pair data structure has string variable used to save the neighbor’s operation and a double variable used to save the neighbor’s value. Then we sort this list by plan’s value in descending order. In each neighbor of the (candidateList), if his value bigger than the (bestValue) we execute this operation even if this neighbor in the tabu list. And updating the bestPlan, currentPlan and the bestValue also we set (IWoutI) with 0. But if his value smaller than the (bestValue), first we check if this neighbor in the tabu list or not. If it in tabu so we have to look for another solution in the candidate list else we execute this operation and increase (IWoutI) value by 1. After checking all neighbors in the candidate list and there are not a solution exist, or the candidate list is empty or (IWoutI) became bigger than (MIWoutI). We delete a place from the currentPlan randomly and return (IWoutI) with 0 value. If this new currentPlan became better the bestPlan we have to update the bestPlan and the bestValue. After doing the (MT) iterations the function returns the bestPlan.

Figure 40 shows how we get the neighbors of the currentPlan.

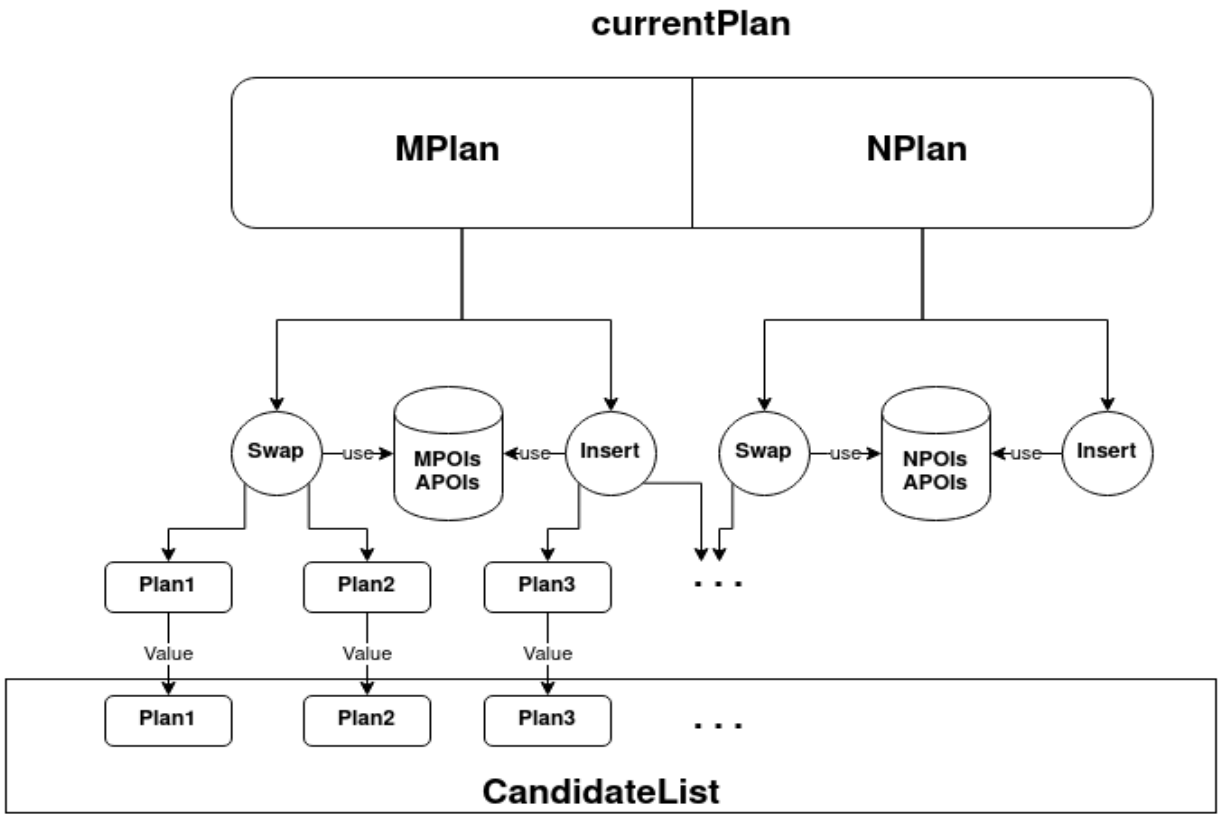


Figure 40 current plan

We use the swap and insert operations to get the neighbors and each output plan represents a neighbor. In swap operation we swap each place not in the plan "outPlace" with each place in the plan "inPlace". Because we working on a plan divided into morning and night sub-plans, in the morning sub-plan we only try swapping places in MPOIs and APOIs as "outPlace". And in night sub-plan we use only NPOIs and APOIs places as "outPlace". In insert operation we insert each place not in the plan "outPlace" in all available index of the plan. And like the swap operation it uses the same place as "outPlace" in each sub-plan. In operation (swap/insert) we do same style to get the neighbor. This style contain 3 main steps:

1. Do

At first we do the operation (swap/insert) on the currentPlan.

2. Evaluate

Then we evaluate the output plan and save this value and the operation in the candidate list.

3. Undo

We return to the currentPlan before doing this operation.

Submit User Profile Setting:

This is the submitted user profile function in setting Page, this page when the user wants to change his information as (image, Name, Password, Address, Favorite Places, Forbidden Places, and Preferences) give user id and data that changed as an argument and return true if data changed and false if data not changed, Figure 72 shows the setting user interface.

```
// Function to submit new user data.
private void submitSettingData() {
    if(mUriImage != null) {...}
    if (!mSettingPassword.getText().toString().equals(mCurrentUserData.getmUserPassword())) {
        String newPassword = mSettingPassword.getText().toString();
        if (CurrentUser != null) {
            CurrentUser.updatePassword(newPassword)
                    .addOnCompleteListener((task) → {
                            if (task.isSuccessful()) {
                                mCurrentUserData.setmUserPassword(mSettingPassword.getText().toString());
                                mUsersTable.child("userPassword").setValue(mSettingPassword.getText().toString());
                            }
                    }); }
    }
    if (!mSettingFullName.getText().toString().equals(mCurrentUserData.getmUserName())) {
        mCurrentUserData.setmUserName(mSettingFullName.getText().toString());
        mUsersTable.child("userName").setValue(mSettingFullName.getText().toString());
    }
    if (!mSettingEmail.getText().toString().equals(mCurrentUserData.getmUserEmail())) {
        mCurrentUserData.setmUserEmail(mSettingEmail.getText().toString());
        mUsersTable.child("userEmail").setValue(mSettingEmail.getText().toString());
    }
    if (!mSettingAddress.getText().toString().equals(mCurrentUserData.getmUserAddress())) {
        mCurrentUserData.setmUserAddress(mSettingAddress.getText().toString());
        mUsersTable.child("userAddress").setValue(mSettingAddress.getText().toString());
    }
    Toast.makeText( context: SettingActivity.this,  text: "Data Submitted", Toast.LENGTH_LONG).show();
}
```

Figure 41 Submit user Profile function

Open Social media To Contact Us:

This to contact and follow ETP in social media like (Facebook, Instagram, tweeter, Pinterest) give application name, application link, and application URL in web as argument and open application if exist in device or website if application not exist in device, Figure 77 shows the contact us user interface.

```
public Intent openSocialApps(String appName, String pageNameInApp, String pageNameInWeb) {
    try { // if app exist.
        getPackageManager().getPackageInfo(appName,  flags: 0);
        Intent appIntent = new Intent(Intent.ACTION_VIEW, Uri.parse(pageNameInApp));
        return appIntent;
    } catch (PackageManager.NameNotFoundException e) { // if app not exist.
        e.printStackTrace();
        Intent appIntent = new Intent(Intent.ACTION_VIEW, Uri.parse(pageNameInWeb));
        return appIntent;
    }

}
```

Figure 42 Open Social media apps function

Add Plan From Dashboard:

This when choose trip from dashboard and you want to add this plan to your plans to execute it; use user id and trip id as arguments and return true if added or false if not, Figure 73 shows the dashboard plan user interface.

```
mUsersTable.child(currentUserData.getmCurrentUserId()).addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        if (dataSnapshot.child("TripsId").exists()) {
            HashMap<String, String> mUserTripsMap = (HashMap<String, String>) dataSnapshot.child("TripsId").getValue();
            ArrayList<String> mUserTripsList = new ArrayList<~>(mUserTripsMap.values());
            if (!mUserTripsList.contains(sTripId))
            {
                mUsersTable.child(currentUserData.getmCurrentUserId()).child("TripsId").push().setValue(sTripId);
                mTripsTable.child(sTripId).child("NumOfExe").push().setValue(currentUserData.getmCurrentUserId());
            }
            else {

            }
        }
        else {
            mUsersTable.child(currentUserData.getmCurrentUserId()).child("TripsId").push().setValue(sTripId);
            mTripsTable.child(sTripId).child("NumOfExe").push().setValue(currentUserData.getmCurrentUserId());
        }
    }
    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {
    }
});
```

Figure 46 add plan from dashboard function

Delete ETP Account:

This is deleted ETP Account Function in profile page, when user want to delete his ETP account appear alert dialog to warn user from delete if submit delete the ETP account will be deleted; use user id as argument and return login page, Figure 67 shows the profile user interface.

```
try {
    mCurrentUser.delete()
            .addOnCompleteListener((task) → {
                    if (task.isSuccessful()) {...}
            });
    mCurrentUserData.setmFirebaseUser(null);
    mCurrentUserData.setmPerfrancesList(null);
    mCurrentUserData.setmFavoritePlacesList(null);
    mCurrentUserData.setmForbiddenPlacesList(null);
    mCurrentUserData.setmTripsIdList(null);
    mCurrentUserData.setmUserRatingApp(null);
    mCurrentUserData.setmUserPhone(null);
    mCurrentUserData.setmUserAddress(null);
    mCurrentUserData.setmUserPassword(null);
    mCurrentUserData.setmUserEmail(null);
    mCurrentUserData.setmUserName(null);
    mCurrentUserData.setmCurrentUserId(null);
    mCurrentUserData.setmUserImage(null);
    mCurrentUserData.setmUserID(null);
    mCurrentUserData.setmRegisterDate(null);
    LogOutFunction(deleteAccountButton.getRootView());
    Intent main = new Intent( packageContext: Profile.this, logIn.class);
    startActivity(main);
    finish();
}
catch (Exception ex){
    Log.d( tag: "GemyDeleteAccountExeption",  msg: "delete account Exeption: " + ex);
}
```

Figure 43 delete ETP Account function

Make Place Favorite or Forbidden:

This is to make a place is favorite or forbidden function, when user want to make a place is favorite or forbidden place based on place information; use place id and user id as arguments and return view, Figure 75 shows the place information user interface.

```
case "Favorite": {
    mUserFavoriteList.remove(mPoiPoiName);
    mUserTaple.child("favoritePlaces").child(mPoiPoiName).removeValue();
    mCurrentUserDataObject.setmFavoritePlacesList(mUserFavoriteList);
    mItemMakeFavoritePlaceButton.setAlpha((float) 0.8);
    mFavoriteItemSelectedImage.setImageResource(R.drawable.ic_no_icon_24dp);
    mItemMakeForbiddenplaceButton.setAlpha((float) 0.8);
    mForbiddenItemSelectedImage.setImageResource(R.drawable.ic_no_icon_24dp);
    mItemSelected = "Null";
    break;
}
case "Forbidden": {
    mUserForbiddenList.remove(mPoiPoiName);
    mUserTaple.child("forbiddenPlaces").child(mPoiPoiName).removeValue();
    mUserFavoriteList.put(mPoiPoiName, mPoiPoiID);
    mUserTaple.child("favoritePlaces").child(mPoiPoiName).setValue(mPoiPoiID);
    mCurrentUserDataObject.setmFavoritePlacesList(mUserFavoriteList);
    mCurrentUserDataObject.setmForbiddenPlacesList(mUserForbiddenList);
    mItemMakeForbiddenplaceButton.setAlpha((float) 0.8);
    mForbiddenItemSelectedImage.setImageResource(R.drawable.ic_no_icon_24dp);
    mItemMakeFavoritePlaceButton.setAlpha((float) 1);
    mFavoriteItemSelectedImage.setImageResource(R.drawable.ic_done_all_black_24dp);
    mItemSelected = "Favorite";
    break;
}
case "Null": {
    mUserFavoriteList.put(mPoiPoiName, mPoiPoiID);
    mUserTaple.child("favoritePlaces").child(mPoiPoiName).setValue(mPoiPoiID);
    mCurrentUserDataObject.setmFavoritePlacesList(mUserFavoriteList);
    mItemMakeFavoritePlaceButton.setAlpha((float) 1);
    mFavoriteItemSelectedImage.setImageResource(R.drawable.ic_done_all_black_24dp);
    mItemSelected = "Favorite";
    break;
}
```

Figure 44 make place favorite or forbidden function

Search by Place Name:

This is search about trips by place name function to filter dashboard trips; these trips uploaded by the other users in dashboard page, to share his trips with other users; use place name as an argument and return trips, Figure 68 shows the dashboard user interface.

```
for (final DataSnapshot mDS:dataSnapshot.getChildren()) {
    try {
        if (mDS.child("TripOnDashBord").getValue(String.class).equals("True")) {
            ArrayList<String> mTripPoisNameList = (ArrayList<String>) mDS.child("PoisNames").getValue();
            mTripPoisNameList.add("All");
            if (mTripPoisNameList.contains(SearchText)) {
                mTripsIdList.add(mDS.getKey());
                mTripsImageList.add(R.drawable.luxor_image + "");
                mTripsNameList.add(mDS.child("PlanNames").getValue(String.class));
                mPostsTimeList.add((mDS.child("fullDuration").getValue(Integer.class) / 60) + " h");
                mTripsRateList.add(mDS.child("TripRate").getValue(String.class));
                mTripsPlaceList.add(mDS.child("TripPlace").getValue(String.class));
                mTripsBudgetList.add(String.format("%.2f", mDS.child("fullCost").getValue(Double.class)));
                mTripsNumOfExeList.add(mDS.child("NumOfExe").getChildrenCount() + "");
                mTripsNumOfCommentList.add(mDS.child("tripCommentsId").getChildrenCount() + "");
            }
        }
    }
    catch (Exception e)
    {
        Log.d( tag: "Dashboard Exeption", e.getMessage());
    }
}
```

Figure 45 search on trips by place name function

Rate Application:

This to rate ETP and leave your message to improve ourselves; use user id, user rate, and user message as arguments and return true if data submit or false if not, Figure 76 shows the rate app user interface.

```
// submit user rating.
buttonRate.setOnClickListener((v) -> {
        mUsersTable.child(mCurrentUserData.getmCurrentUserId()).child("userRatingApp")
                .setValue(answerValue);
        mUsersTable.child(mCurrentUserData.getmCurrentUserId()).child("userRatingMessage")
                .setValue(mUserMessage.getText().toString());
        Toast.makeText( context: RateActivity.this,  text: "Submitted", Toast.LENGTH_LONG).show();
});

// on click go back to profile activity.
rateBackToProfile.setOnClickListener((v) -> { finish(); });
```

Figure 47 rate ETP function

Log Out:

This to log out from ETP account, when user to log out from his account and user another account; use user id as an argument and return login page Figure 67 shows the profile user interface..

```
public void LogOutFunction(View view) {
    switch (mCurrentUserData.getAccountType()) {
        case "Google":{
            GoogleSignIn.getClient( activity: this, new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
                    .build()).signOut().addOnSuccessListener(new OnSuccessListener<Void>() {
                @Override
                public void onSuccess(Void aVoid) {
                    FirebaseAuth.getInstance().signOut();
                    Intent main = new Intent( packageContext: Profile.this, logIn.class);
                    startActivity(main);
                    finish();
                }
            });
        }
        case "Facebook":{
            AccessToken accessToken = AccessToken.getCurrentAccessToken();
            boolean isLoggedIn = accessToken != null && !accessToken.isExpired();
            if (isLoggedIn) {
                LoginManager.getInstance().logOut();
                FirebaseAuth.getInstance().signOut();
                Intent main = new Intent( packageContext: Profile.this, logIn.class);
                startActivity(main);
                finish();
            }
        }
        case "Email":{
            FirebaseAuth.getInstance().signOut();
            Intent main = new Intent( packageContext: Profile.this, logIn.class);
            startActivity(main);
            finish();
        }
    }

}
```

Figure 48 log out from account function

3.4 ACTUAL DATABASE SCHEMA

Using Firebase real time Database

The Firebase real time Database is a NoSQL Database which has a lot of optimizations and features compared with most of relational databases. It includes a flexible rule to define how the data should be structured to provide security and flexibility.

Firebase is a Database stored as JSON objects, which is easier to use than some SQL databases for the way to handle the data like a tree. When you start adding data to your database at automatically creates a node in the existing JSON structure with an associated key.

ETP has 4 json files as declared and following figures:


```
{
  "CommentContent" : "hi",
  "CommentDate" : 1590146754857,
  "CommentId" : "-M7w7U6Giuyz64diZCLt",
  "CommentUserId" : "5eUPJHTyXhfHR7rwDobnJoULh683"
}
```


Figure 49 Comments json file

```
"CloseTime" : "22:00",
"Duration" : [ null, null, "11", "6", "51", "50", "45", "43", "43", "8", "45", "45",
"45", "45", "46", "47", "39", "7", "10", "11", "8" ],
"OpenTime" : "6:00",
"PoiAddress" : "Luxor City, Luxor, Luxor Governorate",
"PoiDescription" : "A complex, the largest part of which was built during the reign
of Amenhotep III and Ramesses II and was completed by Tutankhamun and others",
"PoiID" : "1",
"PoiLatitude" : "25.700413",
"PoiLocation" : "Luxor",
"PoiLongitude" : "32.639054",
"PoiName" : "Luxor temple",
"PoiPhone" : "0",
"PoiPreferences" : [ null, "Photography", "Meditation", "Archaeological" ],
"PoiPrice" : "8.89",
"PoiRate" : "4.7",
"PoiType" : [ null, "Cultural tourism", "Desert tourism", "Pharaonic tourism" ],
"PoiURLImage" : "gs://test-b8daf.appspot.com/PoisImage/1_LuxorTemple.jpg",
"VisitTime" : "30",
"WorkingTime" : "A"
```


Figure 50 Poi's json file


```
"NumOfExe" : {
  "-M7w7BBX_2DccXkKH8cd" : "5eUPJHTyXhfHR7rwDobnJoULh683",
  "-M80ZbgcpgAHOy-p4CyB" : "FlyOqnL6YBa99vX91ItC9McRmGI3"
},
"POICosts" : [ "6.41", "12.83", "5.13", "0.0", "0.0", "4.0", "4.0", "7.0", "0.0", "0.0", "3.0", "8.89" ],
"POIDescribtions" : [ "Describtion", "Describtion", "Describtion", "Describtion", "Describtion", "Describtion",
"Describtion", "Describtion", "Describtion", "Describtion", "Describtion", "Describtion" ],
"POIDurations" : [ "50", "75", "45", "60", "15", "60", "80", "40", "30", "60", "60", "50" ],
"POIEndTimes" : [ "8:50 AM", "10:21 AM", "11:19 AM", "12:25 PM", "12:48 PM", "1:59 PM", "3:38 PM", "5:0 PM", "5:36 PM",
"6:46 PM", "7:57 PM", "8:55 PM" ],
"POIStartTimes" : [ "8:0 AM", "9:6 AM", "10:34 AM", "11:25 AM", "12:33 PM", "12:59 PM", "2:18 PM", "4:20 PM", "5:6 PM",
"5:46 PM", "6:57 PM", "8:5 PM" ],
"PlanNames" : "Visit Luxor",
"PoisId" : [ "6", "5", "10", "13", "7", "15", "16", "3", "17", "18", "20", "1" ],
"PoisNames" : [ "Valley of the Queens", "Valley of the Kings", "Ramesseum", "Tombs of the Nobles", "Colossi of Memnon",
"Howard Carter House", "Banana Island", "Mummification Museum", "Luxor Cornish", "Market tourism Luxor", "Misr Public
Library Luxor", "Luxor temple" ],
"TripDate" : "",
"TripOnDashBord" : "True",
"TripPlace" : "In Luxor",
"TripRate" : "3",
"UploadedBy" : "5eUPJHTyXhfHR7rwDobnJoULh683",
"fullCost" : 51.260000000000005,
"fullDuration" : 625,
"tripCommentsId" : {
  "-M7w7U6HZNgLt1tOhgaF" : "-M7w7U6GiuyZ64dIZCLt",
  "-M86XL2JkMJbw8KyBGS_" : "-M86XL2ExIjc_c7N_4LB"
},
"tripEndTime" : {
  "hour" : 20,
  "min" : 55
},
"tripStartTime" : {
  "hour" : 8,
  "min" : 0
}
```


Figure 51 Trips json file

```
"TripsId" : {
  "-M93VZwZi7pkowezbiZj" : "-M93VZwZi7pkowezbiZi",
  "-M941XBRPzjIeOy5h_2-" : "-M941XBQmlk_lP40NOhl"
},
"activeTrip" : 0,
"favoritePlaces" : {
  "Valley of the Queens" : "6"
},
"forbiddenPlaces" : {
  "Luxor temple" : "1",
  "Temple of Mut" : "19"
},
"preferences" : [ "Photography", "Meditation", "Event" ],
"userAddress" : "Egypt",
"userEmail" : "bolaaskander90@gmail.com",
"userID" : 1,
"userImage" : "gs://test-b8daf.appspot.com/UserImage/1591360787815.jpg",
"userName" : "Bola Askander",
"userPassword" : "123456789",
"userPhone" : "",
"userRatingApp" : "4",
"userRegisterDate" : 1591360701078
```


Figure 52 Users json file

4. TESTING

4.1 EXPECTED TEST SCENARIOS

In this section will present expected scenario to make sure that the end to end functioning of software is working fine or all the business process flows of the software are working fine, and the table *10* explain that:

ID	SEENARIOS	TEST CASES	OUTPUT
1	Open sup page in profile	• Settings page	• Open setting page.
		• Rate page	• Open rate page.
		• Contact us page	• Open contact us page.
		• About us page	• Open about us page.
2	Delete ETP account	• Delete submitted	• Delete ETP account.
		• Delete canceled	• Cancel deleting account.
3	Search in dashboard by place name	• Enter correct place name	• Filter trips by place name.
		• Enter incorrect place name	• Return null from trips.
		• left Empty	• Return all trips.
4	Make place	• If place already favorite	• Remove place from

	favorite		favorite list.
		• If place normal	• Add place to favorite list.
		• If place forbidden	• Remove place from forbidden list. and add to favorite list.
5	Make place forbidden	• If place already forbidden	• Remove place from forbidden list.
		• If place normal	• Add place to forbidden list.
		• If place favorite	• Remove place from favorite list and add to forbidden list.
6	Add plan from dashboard	• If place already added before	• Do nothing.
		• If user own this trip	• Do nothing.
		• If trip no exist in user	• Add trip to user trips.
7	add comment	• if empty cell and continue	• don’t accept and appear message “Enter your comment first”.
		• If user enter any text and continue	• Accept and add the comment to trip comments.
8	Display trip details that in dashboard	• If trip details opened	• Close trip details
		• If trip details closed	• open trip details
9	Social media contacts	• if user have social media app	• open social media app.
		• if user haven’t social media app	• open browser on social media website.
10	Make a call to contact	• if app have permission to make call	• Make a call
		• if app haven’t permission to make call	• user can’t make a call until give app permission.
11	Submit rate	• if user enter comment and choose rate number	• Save comment and ratting number
		• If comment empty and choose rate number	• Save comment and ratting number

		• if user enter comment and don’t choose rate number	• Save comment and default ratting number
		• If comment empty and don’t choose rate number	• Save comment and default ratting number
12	Submit settings	• New password less the 8 characters	• Don’t submit new password.
		• New password more than 7 characters	• Submit new password.
		• New password leaves empty	• Don’t submit new password.
		• User image, user name, and user address is change	• Save changed
		• User image, user name, and user address does not change	• Profile data does not change.
13	Login	• Google Account	• Choose your Google account and login
		• Facebook Account	• Choose your Facebook account and login
		• Email and Password	• Enter your ETP email and password and login
14	Login by Google or Facebook Account	• Submit login	• Login Successful and Open Home page
		• Cancel Login	• Login failed
15	Login by Email and Password	• Enter correct email and correct password.	• Login Successful and open Home page
		• Enter incorrect email and correct password.	• Login failed and appear message wrong email
		• Enter correct email and incorrect password.	• Login failed and appear message wrong password
		• Enter incorrect email and incorrect password.	• Login failed and appear message wrong email and password
		• Empty email and empty password.	• Login failed and appear message please Enter email and password
16	Register	• Enter name, email that	• Login successful and

		contains @ .com, password more than 6 characters, and correct confirm password	open Home page
		• Empty name, email that contains @ .com, password more than 6 characters, and correct confirm password	• Login failed and appear message enter your name
		• Enter name, email that not contains @ .com, password more than 6 characters, and correct confirm password	• Login failed and appear message invalid email
		• Enter name, email that contains @ .com, password less than 6 characters, and correct confirm password	• Login failed and appear message password too short
		• Enter name, email that contains @ .com, password more than 6 characters, and incorrect confirm password	• Login failed and appear message confirm password is wrong
		• All cells are empty	• Login failed and appear message enter your name
17	Forget password	• Enter correct email	• Reset password successful and link to reset password sent to your email
		• Enter incorrect email	• Reset password failed and appear message invalid email
		• Empty cell	• Reset password failed and appear message enter email
18	Upload trip to dashboard	• The trip uploaded before in dashboard	• Remove trip from dashboard
		• The trip not uploaded	• Uploaded successful to

		before in dashboard	dashboard
		• You did not create this trip	• Uploaded failed and appear message you did not create this trip
19	Constrains to make plan	• Enter plan name, start time, end time, budget, preference, forbidden places, and favorite places	• Enter constrain successful and show you best plan according his constrains.
		• Any empty cell from plan name, start time, end time, or budget	• Enter constrain Failed and appear message please enter correct data.
		• Leave all cells empty	• Enter constrain Failed and appear message please enter correct data.
		• Enter plan name, start time, end time, and budget and leave preference, forbidden places, and favorite places empty	• Enter constrain successful and show you best plan according his constrains.
		• Enter end time before start time	• Enter constrain Failed and appear message please enter correct data.
20	Add another place to recommended plan	• Enter start time, end time, place name, and place number	• Add place successful
		• Any empty cell	• Add place failed and appear message enter correct data

Table 10 Expected test scenarios

4.2 UNIT TEST

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
Sign up	Username, user email, password, preferences	Add user to database, open login in page	open login in page	pass
Login in using mail	Email, password	Go to home layout	Go to home layout	pass
Login in using google	Userauth to login using google	Go to home layout	Go to home layout	pass
Login in using Facebook	Userauth to login using Facebook	Go to home layout	Go to home layout	pass
Reset password	user email	Reset password	Open login page	pass
Select forbidden	List of forbidden names to the function	Dialog contains list of forbidden	Dialog contains list of forbidden	pass
Select favorite	List of favorite names to the function	Dialog contains list of favorites	Dialog contains list of favorites	pass
Read json file	Json file have data about poi	-	Return String variable store value about each poi	pass
Poi's data	String variable have data about poi	-	Store in array each data about poi	pass

Table 11 Unit test of login and register class

Home class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
SetUserData	-	user's name and image	user's name and image	Pass
GetUserPlans	-	All user's plans in database	user's plans	Pass
ViewPlansData	-	Show plan's information (place, name, …)	plan's information	Pass
ViewPlanPlaces	-	View plan's places information (place name, place min cost,…)	plan's places information	Pass
bottom_nav_home	-	Move to any other pages from home page by navigation bar	Moved to other pages	Pass
ekko	-	Open chatbot layout and run Ekko class	Chatbot opened	Pass

Table 12 Unit test of home class

Constrains class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
bottom_nav_home	-	Move to any other pages from constrains page by	Moved to other pages	Pass

		navigation bar		
SetUserData	-	Set user's name and image	user's name and image	Pass
ekko	-	View chatbot layout and run Ekko class	Chatbot opened	Pass
getItemsFromString	text, list	Break the string into items and put them into the list	List items	Pass
selectItems	-	Open dialog to allow user to select his preferences	Dialog opened	Pass
selectforbidden	-	Open dialog to allow user to select his forbidden list	Dialog opened	Pass
selectFavorite	-	Open dialog to allow user to select his favorite list	Dialog opened	Pass
selectHour	-	Open dialog to allow user to select the hour in the trip start and end times	Dialog opened	Pass
selectTimeType	-	Open dialog to allow user to select the type (AM/PM) in the trip start and end times	Dialog opened	Pass

selectMin	-	Open dialog to allow user to select the minute in the trip start and end times	Dialog opened	Pass
CF12T24	hour, type	Hour integer value in 24-time format	Hour in 24-format	Pass
apply	-	Send data to plan review page and open it if the data valid else show error message	Plan review page opened/ error message	Pass

Table 13 Unit test of constrains class

showPlan class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
bottom_nav_home	-	Move to any other page from constrains page by navigation bar	Moved to other pages	Pass
SetUserData	-	Set user's image	user's image	Pass
readJSONFromAsset	context	Read the json file	Json file readed	Pass
CF24T12	hour, min	A string of the time in 12-time format	String time in 12-format	Pass
CF12T24	hour, type	Hour integer value in 24-	Hour in 24-format	Pass

		time format		
addItems	-	Open dialog to allow user to add new place in plan	Dialog opened	Pass
getID	Place name	Place ID by his name	ID	Pass
selectPlaceAdd	-	Open dialog to allow user to choose the place	Dialog opened	Pass
selectPlace	-	Open dialog to allow user to choose the place in min time between 2 places part	Dialog opened	Pass
selectPlaceNum	-	Open dialog to allow user to choose the index	Dialog opened	Pass
selectHour	-	Open dialog to allow user to select the hour in the place start and end times	Dialog opened	Pass
selectMin	-	Open dialog to allow user to select the min in the place start and end times	Dialog opened	Pass

Table 14 Unit test of show plan class

Data class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
userData	-	User object	User object	Pass

		has user's data		
tripData	-	Trip object has trip's data	Trip object	Pass
POIsData	User's preferences and favorite list	List of POI objects have places information	List of POI objects	Pass

Table 15 Unit test of data class

Ekko class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
POIsData	User's preferences and favorite list	List of POI objects have places information	List of POI objects	Pass
readJSONFromAsset	context	Read the json file	json file readed	Pass
start	Integer flag	Chatbot sends start message	Message sent	Pass
about	-	Chatbot sends about the application message	Message sent	Pass
profilePage	-	Chatbot sends profile page information message	Message sent	Pass
settings	-	Chatbot sends settings page information message	Message sent	Pass
favoriteList	Integer flag	Chatbot sends favorite list information message	Message sent	Pass
forbiddenList	Integer flag	Chatbot sends forbidden list information	Message sent	Pass

		message		
preferences	Integer flag	Chatbot sends preferences information message	Message sent	Pass
dashboardPage	-	Chatbot sends dashboard page information message	Message sent	Pass
post	-	Chatbot sends dashboard post information message	Message sent	Pass
dashboardmore	-	Chatbot sends more dashboard post information message	Message sent	Pass
poiInfoPage	-	Chatbot sends Places page information message	Message sent	Pass
poiInfo	-	Chatbot sends Place information message	Message sent	Pass
home	-	Chatbot sends home page information message	Message sent	Pass
planInfo	-	Chatbot sends home's plan information message	Message sent	Pass
planMoreInfo	-	Chatbot sends more home's plan information	Message sent	Pass

		message		
constrainsPage	-	Chatbot sends constrains page information message	Message sent	Pass
st_enTime	Integer flag	Chatbot sends start and end times information message	Message sent	Pass
budget	-	Chatbot sends trip budget information message	Message sent	Pass
showplanPage	-	Chatbot sends Plan review page information message	Message sent	Pass
PlanProperties	-	Chatbot sends plan properties message	Message sent	Pass
planPoiInfo	-	Chatbot sends plan's places information message	Message sent	Pass
modify	-	Chatbot sends modify plan information message	Message sent	Pass
deletePoi	-	Chatbot sends deleting place from plan information message	Message sent	Pass
addPoi	-	Chatbot sends inserting place from plan	Message sent	Pass

		information message		
choosePlace	-	Chatbot sends choosing place information message	Message sent	Pass
choosePlaceN	-	Chatbot sends choosing place number information message	Message sent	Pass
EkkoServices	-	Chatbot sends his services information message	Message sent	Pass
SPs	-	Chatbot opens 2 places dialog and sends min time message	Message sent	Pass
Esound	-	Chatbot mutes/unmutes message sound	Sound muted/unmuted	Pass
EpoiInfo	-	Chatbot opens a places dialog and sends place information message	Message sent	Pass
EkkoButton	-	Option button layout	Option button layout	Pass
userMessage	text	User message layout	User message layout	Pass
reply	options, message, imgurl	Add user's and chatbot messages	user's and chatbot messages added	Pass

Table 16 Unit test of ekko class

FullPlan class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
evaluatePlan	Satisfaction factor weight, travel time weight, waste time weight,	If plan is valid return his value else return -1	Value/-1	Pass
evaluateSubPlan	Plan, current Time, trip	Return True if plan is valid else return False	True/False	Pass
getPlanD	Start Time List, end Time List, Names List, cost List, duration List, description List, IDs List, image URL List	Take add plan places information in input lists	input lists updated	Pass

Table 17 unit test of full plan class

Plan class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
getLastPOI	-	Return last POI object in plan	last POI object in plan	Pass
getLastNode	-	Return last Node object in plan	last Node object in plan	Pass
makeCalculations	Start Time, end Time, full cost, mPlan	Make all Calculations in current sub plan to	Calculations made	Pass

		get (fullCost, wasteTime, travelTime, value)		
insert	poi, index, from, to,PlanStructure	insert new POI in plan in specific index	POI inserted	Pass
swap	poi, id, from, to	Swap poi with poi in plan by his id	POI swapped	Pass
delete	index, tabu List, freq,PlanStructure	Delete poi in plan by his index	POI deleted	Pass

Table 18 Unit test of plan class

Planner class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
getPlan	MT, Tabu Freq, SFW, TTW, WTW, MIWoutI, data	Best plan	Best plan	Pass
DoSolution	name,current plan, prepare, tabu List, TabuFreq	Do an operation on the current plan	Operation did on the current plan	Pass

Table 19 Unit test of planner class

Prepare class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
makeSubPlan	plan, current	Sub plan	Sub plan	Pass

	Time, POIs, time End, trip, mPlan, skip			
canInsertLast	plan, poi, trip, current time, mPlan, skip	True if place can be inserted at last position of the plan else False	True/False	Pass

Table 20 Unit test of prepare class

TabuList class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
update	MPOIs, NPOIs, APOIs	Update tabu list	Tabu list updated	Pass
Isearch	Id, index	True if insert operation in tabu list else False	True/False	Pass
Iinsert	Id, index, freq	Add insert operation in tabu list	Operation added in tabu list	Pass
Ssearch	Id1, id2	True if swap operation in tabu list else False	True/False	Pass
Sinsert	Id1, id2, freq	Add swap operation in tabu list	Operation added in tabu list	Pass
Dinsert	Poi, freq	Add delete operation in tabu list	Operation added in tabu list	Pass

Table 21 Unit test of tabu list class

Time class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
isIn	Start Time, duration, end Time	Check if start time + duration <= end time True if yes False if not	True/False	Pass
compare	Time	Check if this Time before or equal argument time true if yes otherwise false	True/False	Pass
add	duration	Add duration to current Time and if result passed that day (24) return false otherwise true	True/False	Pass
substract	Time1, Time2	Substract between 2 times and return -1 if time1>time2 otherwise the result	Result/ -1	Pass
max	Time1, time2	maximum time	maximum time	Pass
min	Time1, time2	minimum time	minimum time	Pass

Table 22 Unit test of time class

Profile Class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
OpenSubPage	nameOfNewSubPage To the OpenSubPage function	Open another page	Open another page	pass
OpenShare	-	Open dialog to share the app	Open dialog	pass

SetUserData	UserId to the SetUserData function	User picture and user name	Picture and name	pass
SignOut	userAuth to the DeleteAccount function	Open login page	Open login page	pass
DeleteAccount	userAuth to the DeleteAccount function	Delete user authentication and open login page	Open login page	pass
BottomNavProfile	-	Open another page	Another page opened	Pass

Table 23 Unit test of profile class

Settings Class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
chooseImage	-	Open mobile Storage	Picture	pass
setDataInDaialog	daialogType, listOfItems to the setDataInDialog function	Dialog contains list of places	Dialog contains list of places	pass
fillListByData	List contains name of places to the function	-	-	pass
SelectItems	List of preferences names to the function	Dialog contains list of preferences	Dialog contains list of preferences	pass
deletePriviousImage	imageUrl to the function	Delete image	deleteimage	pass
SubmitData	-	Submit new data	Submit new data	pass
getFileExtension	Picture to the	Get picture	Picture	pass

	function	extension	extension	
SetUserData	UserId to the SetUserData function	User picture and user name	Picture and name	pass
goBack	-	Open profile page	Profile page opened	pass

Table 24 Unit test of settings class

rate Class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
SubmitRate	newRate, RateMessage to the function	Save new rate and message	Save new rate and message	pass
SetRate	PriviousRate to the function	-	-	pass
goBack	-	Open profile page	Profile page opened	pass

Table 25 Unit test of rate class

Contact us Class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
openSocialApps	appName, pageName, pageNameInWeb to the function	Open App or Website	Open App or Website	pass
sendEmail	EmailAddress to the function	Open email application	Open email application	pass
makeCall	phoneNumber to the function	Make a call	Make a call	pass
goBack	-	Open profile page	Profile page opened	pass

Table 26 Unit test of contact us class

Dashboard Class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
SearchByPlace	PlaceName to the SearchByPlace function	Filter the plans by place name	Filter the plans by place name	pass
SetUserData	UserId to the SetUserData function	User picture and user name	Picture and name	pass
fillVariableTripData	-	Initialization variables	Initialization variables	pass
ShowPlans	listOfPlans to the function	Appear List of Plans	Appear List of Plans	pass
BottomNavDashboard	-	Open another page	Another page opened	pass

Table 27 Unit test of dashboard class

Post in dashboard Class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
fillTripData	-	Show trip data	Trip data appeared	pass
SetComment	Commentcontent, userId, to the function commentdate to the setCommrnt function	Comment appear in comments area	The comment appeared	pass
AddExistTrip	userId, TripId to the addExitTrip function	Trip add to user trips	The trip added	pass
goBack	-	Open dashboard page	Dashboard page opened	pass

Table 28 Unit test of post in dashboard class

Places Class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
setUserDataFunction setUserData	UserId to the SetUserData function	User picture and user name	picture and user name	pass
ShowPlaces	listOfPlaces to the function	Appear List of Places	List of Places appeared	pass
FillPlacesData	-	Initialization variables	Variables initialized	pass
BottomNavProfile	-	Open another page	Another page opened	pass
OpenPlace	PlaceId to the function	Open new page contains place information	Place information opened	pass

Table 29 Unit test of places class

Place Information Class

Function Name	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
getPlaceInformation	PlaceId to the function	Initialization variables	Initialization variables	pass
fillPlaceData	PlaceId to the function	Place data appear in page	Data appeared	pass
getItemType	userId to the function	Place type selected	type selected	pass
makePlaceFavorite	PlaceId, UserId to the function	Change type of place	Type changed	pass
makePlaceForbidden	PlaceId, UserId to the function	Change type of place	Type changed	pass
makePlaceNormal	PlaceId, UserId to the function	Change type of place	Type changed	pass
goBack	-	Open places page	Places Page opened	pass

Table 30 Unit test of place information class

4.3 FUNCTIONAL TEST

Using 20 different places in the system, we made more than 70 tests for getting user's plans. The maximum time the algorithm needed to do his operation is 3 seconds. There are many variables in the algorithm. So that we made many tests for most of these variables to test getting plan functionality. Most of these variables are:

1- Maximum number of iterations
2- Generating the initial plan
3- Waste time weight
4- Travel time weight
5- Satisfaction factor weight

In the next sections we will explain how we tested each of these variables.

1) Maximum number of iterations

We made 10 tests on (MT) variable. In each test, we change the value of (MT) only. The plan constrains for each test is constant. The (SFW) is a constant with value 1. The (TTW) is a constant with value 3. The (WTW) is a constant with value 2. The (MIWoutI) is a constant with value 10. The (TabuFreq) is a constant with value 3.

Figure 53 shows 10 tests on the (MT) with their final plan evaluation value.

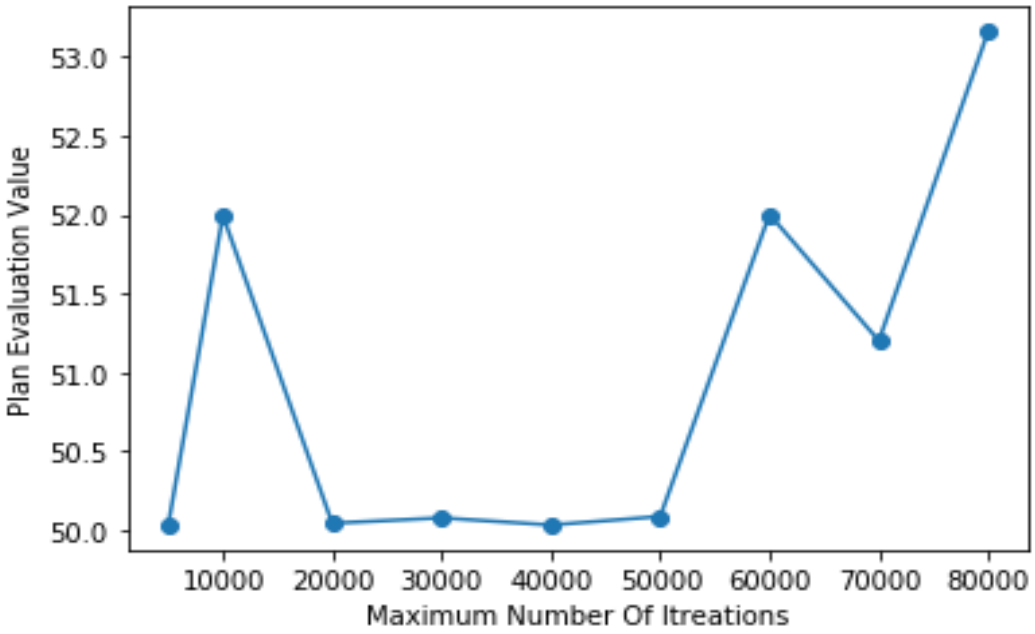


Figure 53 tests on the (MT) with their final plan evaluation value

Figure 54 shows 10 tests on the (MT) with algorithm execution time in millisecond.

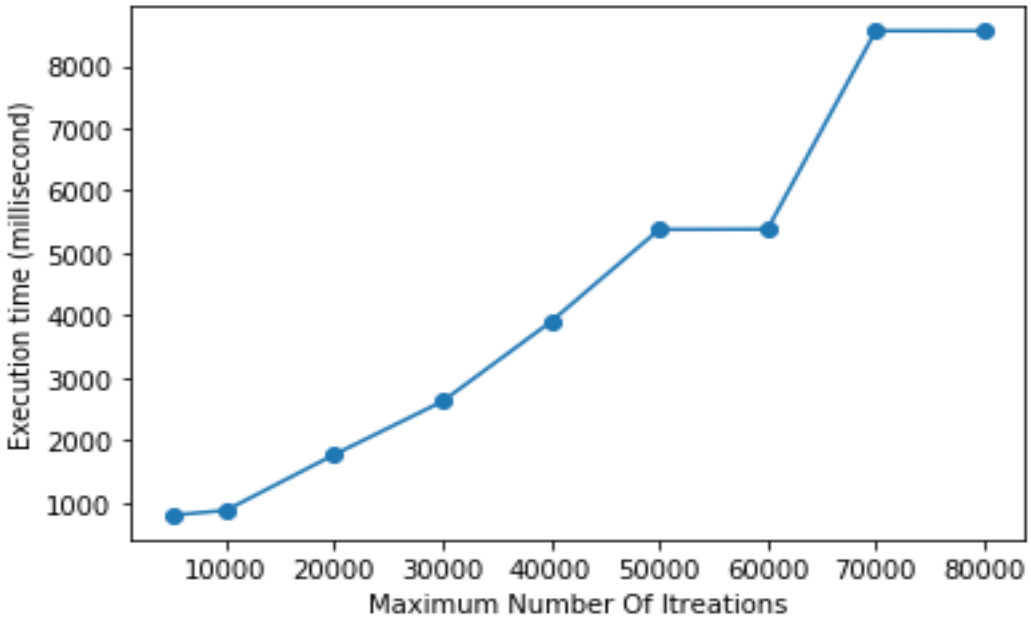


Figure 54 tests on the (MT) with algorithm execution time in millisecond

2) Generating the initial plan

To generate an initial plan by using prepare class, we have 2 options:

A. Initial plan with Reordering POIs: After getting the POIs of the system, we change their order randomly before be used to generate the initial plan.

B. Initial plan without Reordering POIs: After getting the POIs of the system, we send them directly to generate the initial plan.

We made 10 tests in each option. In each test (SFW), (TTW), (WTW), (MIWoutI), (TabuFreq) and (MT) are constants with values 1, 3, 2, 10, 3 and 10000. Plan constrains change in each test but each option take the same constrains. For example, in test 1 with and without reordering options use the same plan constrains but these constrains will be identically changed in test 2.

Figure 55 shows 10 tests on generating plan with reordering. Each line represents a test and has 2 marker points which refer to the initial plan evaluation value and the final plan evaluation value. The x-axis represents the execution time for each test in millisecond. The y-axis represents plan (initial/final) evaluation value. For

example the blue line represents the first test. The initial plan evaluation value in this test was 38 and the final plan evaluation value is 54. The execution time for this test is 1479.

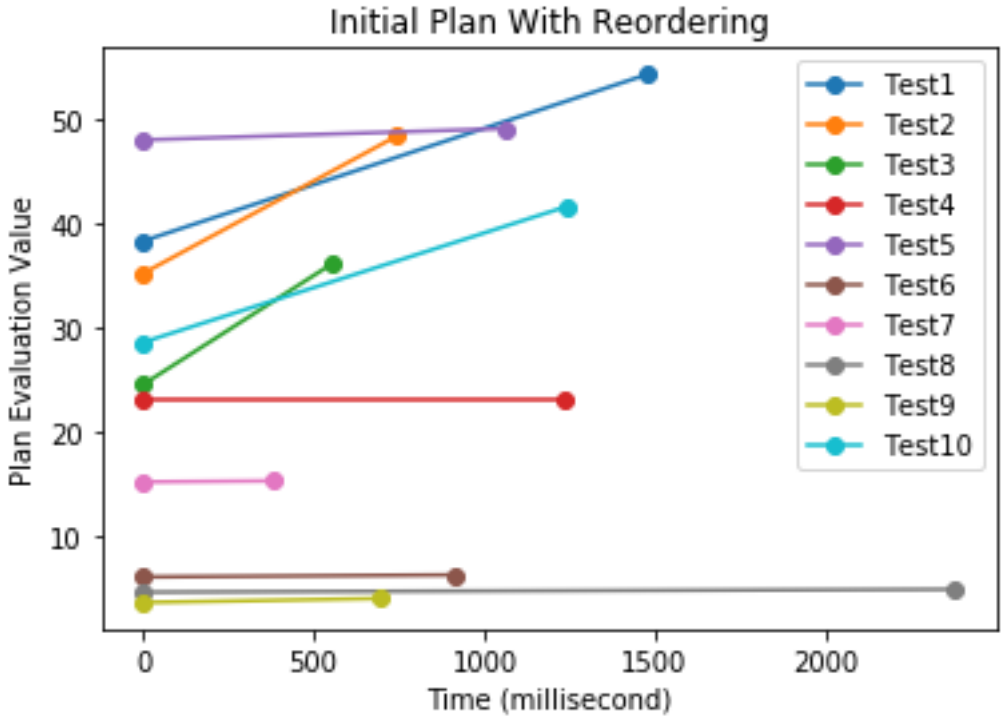


Figure 55 The execution time for this test is 1479

Figure 56 shows 10 tests on generating plan without reordering.

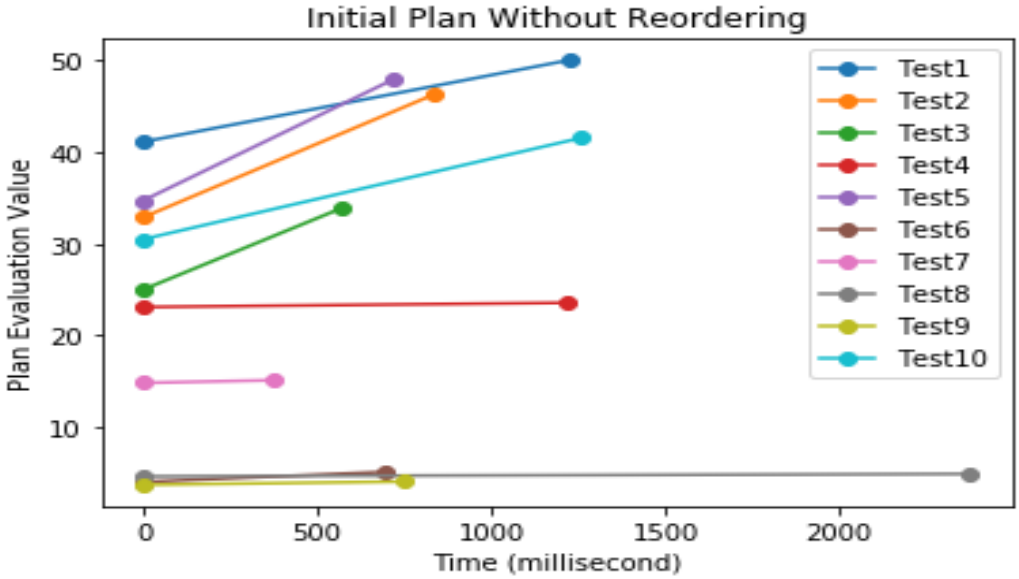


Figure 56 tests on generating plan without reordering

Figure 57 shows a comparison between, Initial plan with Reordering and Initial plan without Reordering in the same test and the same plan constraints.

Figure 57 Initial plan with Reordering and Initial plan without Reordering

In waste time, satisfaction factor, and travel time weights, we made 10 tests for each of them. Plan constraints, (MIWoutI), (TabuFreq), (MT) are constants. These weights will not always make the expected change because of many reasons such as in every 10 tests, we make 2 of these weights constant and 1 changes in each iteration. These weights depend on each other in the evaluation so that they need a separate optimization algorithm to find their best values. In general, the tabu search works randomly and because of this randomness, the mistake percentage exists.

3) Waste time weight

In (WTW) 10 tests, (TTW) and (SFW) are constants with values (3,1).
Figure 58 shows the tests waste time weights and the final plan waste times in minutes.

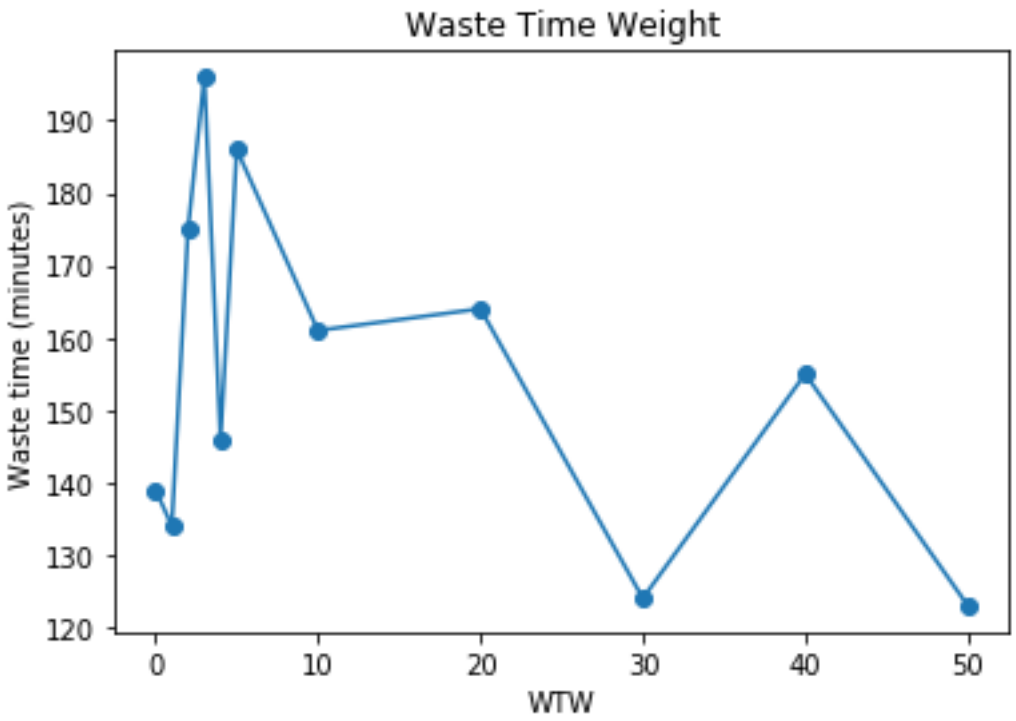


Figure 58 the tests waste time weights and the final plan waste times in minutes

4) Travel time weight

In (TTW) 10 tests, (WTW) and (SFW) are constants with values (2,1).

Figure 59 shows the tests travel time weights and the final plan travel times in minutes.

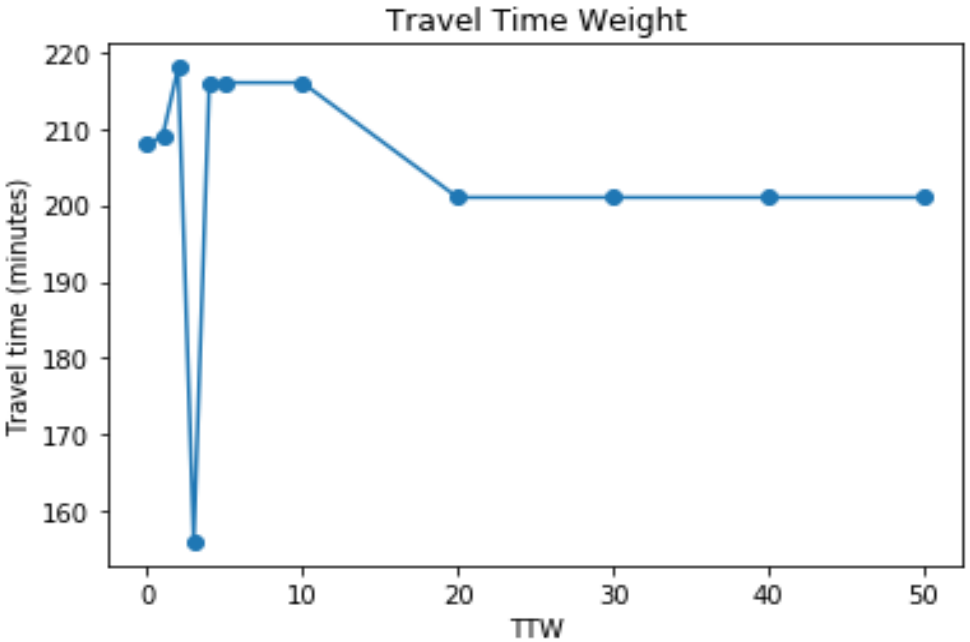


FIGURE 59. the tests travel time weights and the final plan travel times

5) Satisfaction Factor weight

In (SFW) 10 tests, (WTW) and (TTW) are constants with values (2,3).

Figure 60 shows the tests satisfaction factor weights and the final plan satisfaction factor value.

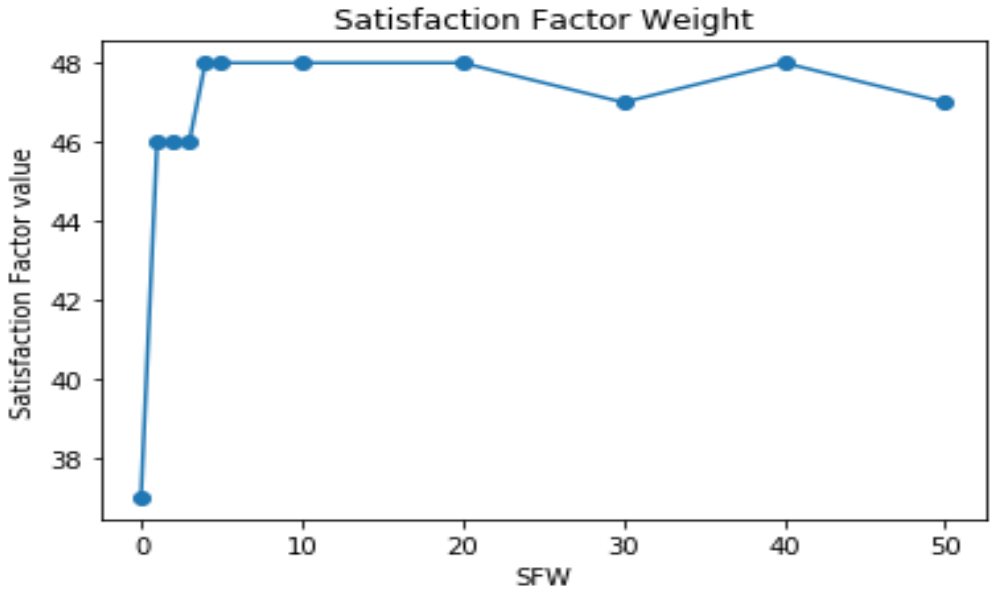


FIGURE 60. the tests satisfaction factor weights and the final plan satisfaction

4.4 USABILITY TEST

We brought random 20 users to test the project and observing how real people are already using it are effective ways to determine whether your visitors: understand how the project works and don't get 'lost' or confused can complete the main actions they need to don't encounter usability issues or bugs have a functional and efficient experience notice any other usability problems

Figure 61 Usability Test

5. DEPLOYMENT OF THE SYSTEM

- If you want to publish a paid app or plan to sell in-app purchases, you need to create a payments center profile, i.e. a merchant account. To do so, sign in to your Play Console.
- Now that you have set up your Play Console, you can finally add your app. to do so, navigate to the 'Create Application', select your app's default language, and type in a title for your app.
- Before you can publish your app, you need to prepare its store listing. These are all the details that will show up to customers on your app's listing on Google Play. The information required for your store listing is divided into several categories:
    - Your app's title and description should be written with a great user experience in mind.
    - Use the right keywords, but don't overdo it. Make sure your app doesn't come across as spam-y or promotional, or it will risk getting suspended on the Play Store.
    - You can add screenshots, images, videos, promotional graphics, and icons that showcase your app's features and functionality.
    - There are specific requirements for each graphic asset that you upload, such as the file format and dimensions.
    - can add translations of your app's information in the store listing details, along with in-language screenshots and other localized images.
    - There are various categories for each type of app available on the Play Store. Pick the one your app fits into best.
    - You must add a URL linking to your privacy policy in your store listing and within your app. Make sure the link is active and relevant to your app.
    - Now that you have prepared the ground to finally upload your app, it's time to dig out your APK file.
    - The Android Package Kit (or APK, for short) is the file format used by the Android operating system to distribute and install apps. Simply put, your APK file contains all the elements needed for your app to actually work on a device.
    - After you're done, press Save.

6. LIMITATION OF THE SYSTEM

- We wanted to add new features in ETP, but there are a group of obstacles, which is adding other places in the ETP other than those that exist, but there are a set of problems the most important is that there is no reliable source for these other places and also there is not enough information about them that you would like if added That adversely affect the performance of the ETP.

- Wanted to implement this program in Luxor and Aswan and added another cities, because Luxor and Aswan related to each other in the eyes of the tourists and the proximity of the distance between them, but as we mentioned earlier obstacles that made us not implement it.

- Wanted to show the Pois of trip in Google Maps to make it easy to user to go this places and also use of Google Maps has another set of features, But we did not want to risk using this feature because it is temporarily free, and we did not want to risk adding it to the project and upon delivery of the project this free period ends, and we do not have the ability now to buy it completely.

- Made the user choose the Start and end positions of trip, but we did not implement that because we needed Google Maps, and also that would make the algorithm it would be very complicated and it could be difficult to implement it in practice.

7. CONCLUSION AND FUTURE WORK

ETP has made a strong effort to automate travel planning and provide a reliable service that collects all the information tourists need to prepare their trips. In fact, ETP successfully overcome one of the main challenges for tourists, finding information in many sources, by making everything available in one Android app.
ETP provides a great service for the tourist companies, which through this project it is possible to create a set of plans that are compatible with a group of tourists they are deal with them, ETP could work as a standalone application to provide the best tourist service such as taxi companies and companies that related to tourist industry,The system facilitates to increase the tourist to visit Luxor, which leads to an increase in the interior of these cities and the provision of work opportunities through it.
In future work, this algorithm may be improved in getting the plan. Adding a system used to check if the user's changes on the plan are valid or not. Adding more cities and make ETP At the level of all Egypt. Having a budget to use some services to increase the functionality of ETP that we mention them on Limitation on the system. Changing the chatbot from static to dynamic using (NLP, AI, ML,...). Improving the dashboard by making a recommendation system.

8. REFERENCE

1. Kadri Sylejmani and Agni Dika. Solving touristic trip planning problem by using taboo search approach

2. Aldy Gunawan · Hoong Chuin Lau. Kun LuA Fast Algorithm for Personalized Travel Planning Recommendation 2016

3. Hsiu-Sen Chiang Tien-Chi Huang User-adapted travel planning system for personalized schedule recommendation. 2015

4. Merlinda Sumardi, Jufery, Frenky, Rini Wongso*, Ferdinand Ariandy Luwinda. “TripBuddy” Travel Planner with Recommendation based on User‘s Browsing Behaviour 2017