TED University

Department of Computer Engineering

CMPE 492 Senior Project II

Plan My Day

Low-Level Design Report

Project Group Members: Gökalp Fırat
Nehir Yiğit Karahan

Supervisor: Tayfun Küçükyılmaz

February 25, 2019

Contents

Low-Level Design Report

Plan My Day

1 Introduction

Plan My Day is being developed to help people with managing their daily plans. People cannot manage their time efficiently due to some reasons like not having enough knowledge about the city they are living in or locations of the places where they want to be.

We are going to develop a mobile application which is PMD (Plan My Day) as a solution to these problems mentioned above. Users tell their plans to the application by writing either speaking and the most efficient path for them is decided by our software.

1.1 Object Design Trade-offs

Developing an A.I. assistant application is different from developing simple mobile applications. Design principles are changed for our application too. The application focus will be less on functionality but more on usability and robustness.

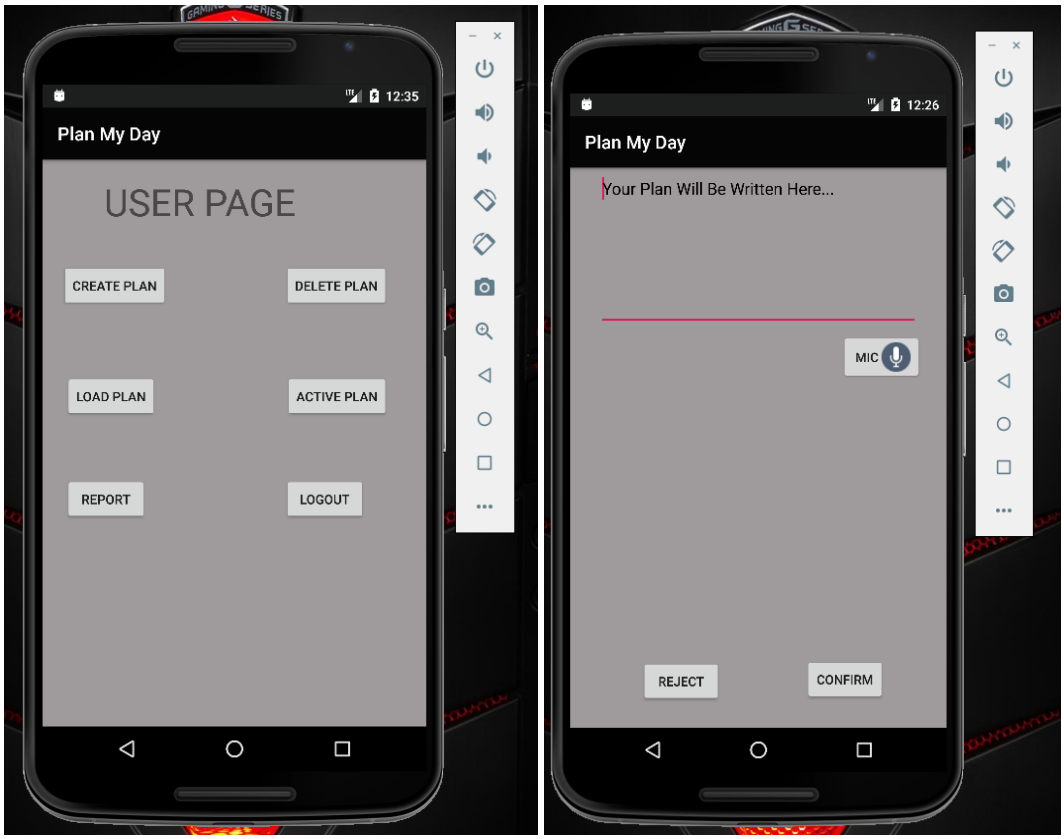
1.1.1 Functionality versus usability

At the beginning of mobile application era, developers prefer functionality to show how far device can function. However in today's applications users are mostly prefer higher usability ones. The main reason of it is the popularity of smartphone usage spread through every age. Therefore an application should usable for all ages. Too young and too old people may be new at smartphones and more functional applications can be seen as complex. Thus, usability is of higher priority than functionality.

1.1.2 Cost versus robustness

One of the main principles of an A.I. assistant application is minimizing errors. A.I assistant applications are created to facilitate people's life. They use text or voice type inputs for their utility functions and they need to interpret user's requests truly. In order to increase success rate of this interpretation developers must carry about erroneous inputs. Everyday A.I. applications are getting better at interpretations because users expect A.I. to function like real life human assistant. That's why robustness has higher priority than cost.

1.2 Interface Documentation Guidelines



User's will open app and login with their credentials if they have created an account earlier. If not, they need to create one or continue as guest. After that login/register process user allow some permission such as GPS for application to work properly. User can create, delete or load a plan. Plan creating methods are by voice or by text. These plans are interpreted by the application and find places. To create a route for a plan user need to activate it and then they can use Google Maps to navigate through their plan.

Example interface usage scenario:

1) User opens our app to plan his/her day according to distances.
2) User allows GPS permission to our App.
3) User click on + icon to add plans to his/her list.
4) Based on user's preference app will open a text box or microphone button.
5) User writes his/her plan to the text box or tell his/her plan to the microphone by pressing microphone button.
6) User will use app's manual to learn how to use conjunctions.
7) User's plan consists places that he/she needs to go according to his/her plan.
8) After the plan is told, app will show a preview of the places and the plan.
9) User checks if it is true or correct missing parts.
10) After confirming, app navigate to the homepage that shows user's route.
11) User chooses to be navigated, so that app uses Google Maps to navigate the user to closest place according to plan.
12) User add new places or plans with + icon.
13) If he/she decides to cancel plan, they can remove with thrash icon located on the right side of the plan.

1.3 Engineering Standards

The engineering approach of our application is UML-based. Unified Modeling Language was evolved from the integration of the three different modeling approaches of Booch, OOSE and OMT seemed to be a promising approach for system modeling. ”Since the early integration efforts, the UML became the “lingua franca” of (object-oriented) software engineering (Object Management Group, 2005). A prominent feature of UML is that it provides a set of aids for the definition of domain-specific modeling languages (DSL) – so called extension mechanisms. Moreover the newly defined DSLs remain UML-compliant, which allows the use of all UML features supplemented, e.g., with Web specific extensions.”

Instead of using other modeling techniques, the reasons for the choice of the IML as a standard are the flexibility provided by the extension mechanisms. Model-driven Engineering combined with UML, as already used in software engineering, could provide abstraction and automation for mobile software development. It has different representations such as graphical and textual. These are best to present our development process from start to end. UML is best for representing object oriented software and since our application will be written in an object oriented language Java, UML standards are best approach for our application.

1.4 Definitions, acronyms, and abbreviations

The following definitions will be used:

UML: is a standardized modeling language enabling developer to specify, visualize, construct and document artifacts of a software system. Thus, UML makes these artifacts scalable, secure and robust in execution. UML is an important aspect involved in object-oriented software development.

Robustness: is the ability of a computer system to cope with errors during execution and cope with erroneous input.

Variable: a class variable is an important part of object-oriented programming that defines a specific attribute or property for a class and may be referred to as a member variable or static member variable.

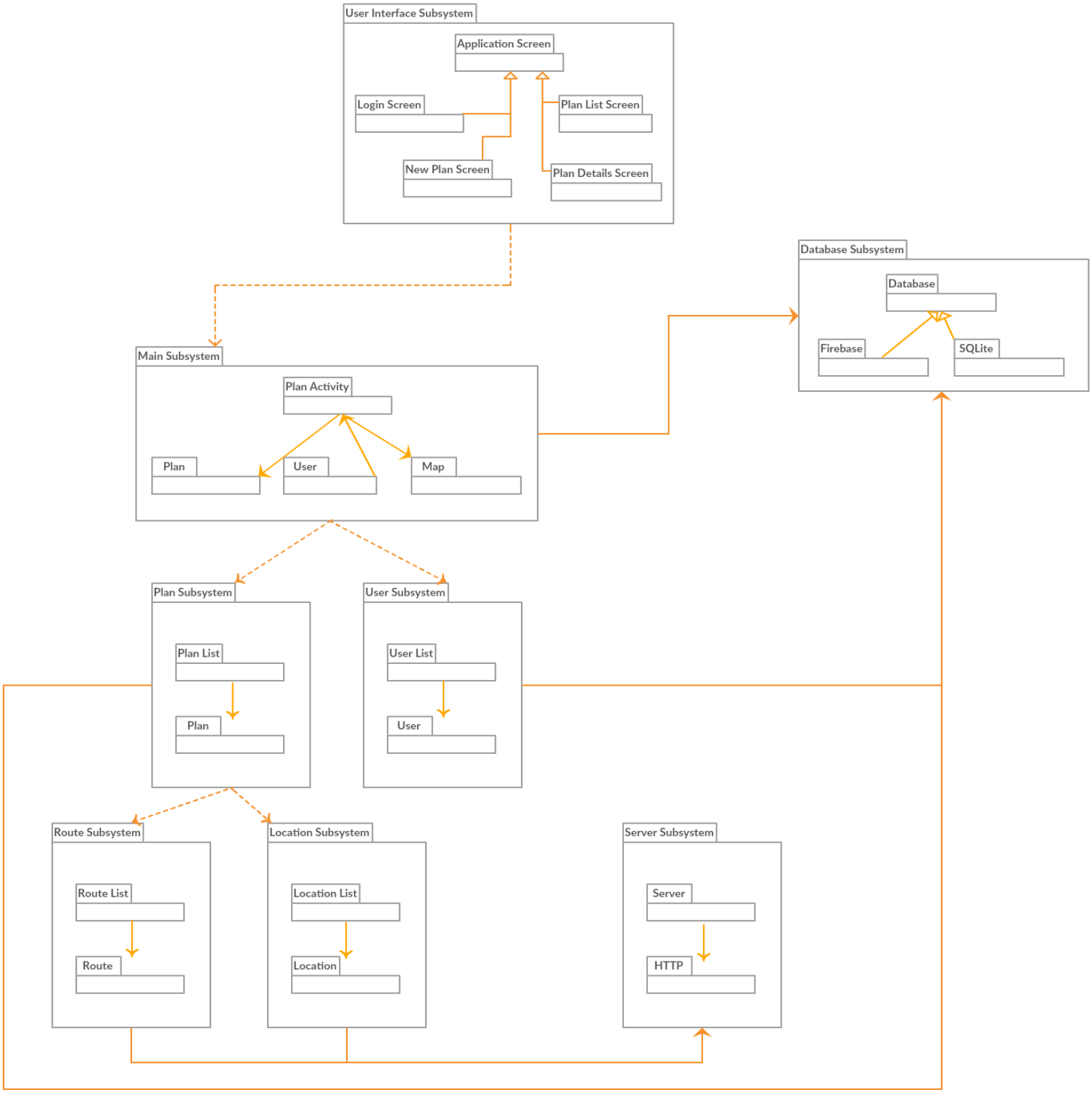
Method: a method in object-oriented programming is a procedure associated with a message and an object. An object is mostly made up of data and behavior, which form the interface that an object presents to the outside world. A method is Java programming sets the behavior of a class object.

Interface: An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types.

CRUD: In computer programming, create, read, update, and delete (CRUD) are the four basic functions of persistent storage. Alternate words are sometimes used when defining the four basic functions of CRUD, such as retrieve instead of read, modify instead of update, or destroy instead of delete.

2 Packages

In overview, the application will consist of several packages. This results in the following subsystem decomposition figure below. Each subsystem can be considered as an individual package.

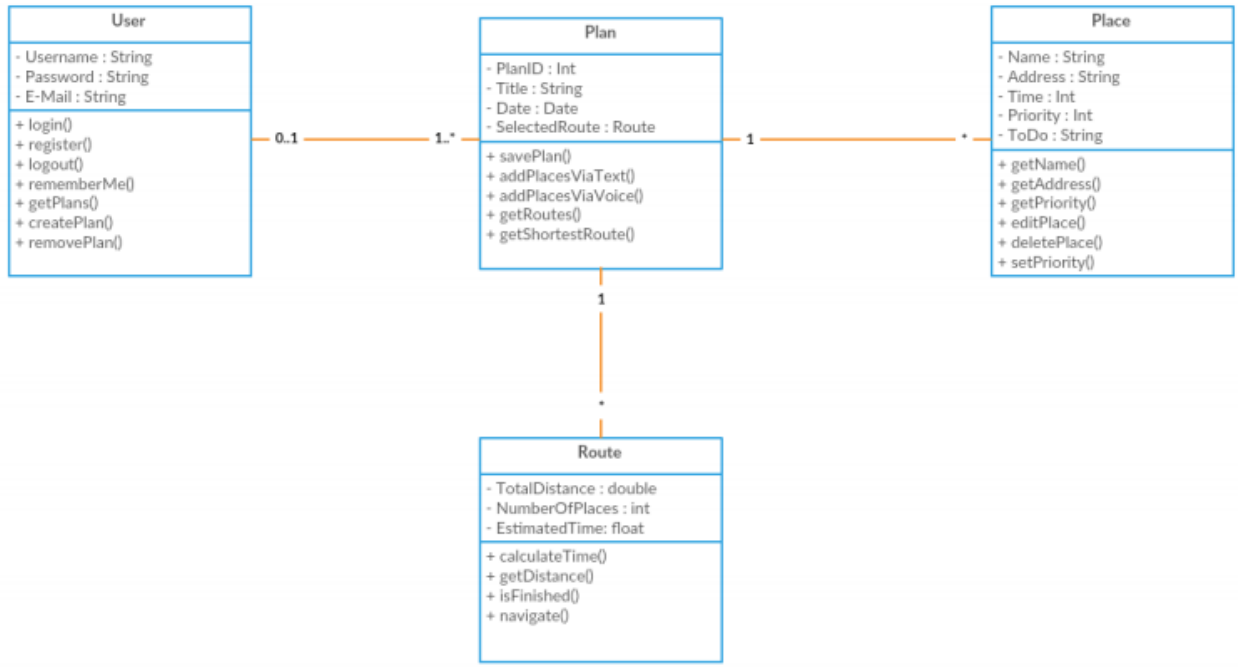


The *User Interface* package is responsible for all interactions between the users and relatives. All entry data, modification will be handled directly through the user interface package. This package decomposed into set of pages loaded and displayed in device.

The *Main* package is responsible for all interactions between UI system and *Plan*, *User* packages. Plan, User and Map classed are combined to create *Main* package. It is also responsible for interactions between the User and Plan classes to *Database* package.

The other packages *Plan*, *User*, *Route and Location* are responsible for all interactions that is related to them and handled through by their List class. Also *Plan* and *User* are responsible for their classes/objects interaction with *Database* package. *Route* and *Location* packages are responsible for their classes/objects interaction with *Server* package.

3 Class Interfaces



Our application will use *User*, *Plan*, *Route* and *Place* classes. These classes will be used at the entire A.I. algorithms. Also these classes have their own packages which manage the interactions between other classes and packages. These classes have different variables and methods according to their functionality.

3.1 User Class

User class will have *Username*, *Password* and *E-mail* variables. These are the identifiers for specific user and username will be unique key that can't be same with another user. Password will be stored as hash for security purposes and will be used at login. These variables will be defined at registration stage.
This class will have common methods for everyday user systems such as *login*, *register*, *logout*, *rememberMe* and some specific methods for our application like *getPlans* and *createPlan*.

3.2 Plan Class

Plan class will have a identifier variable *PlanID* and other common variables *Ttle,Date.* There is a specific variable which points a route which is another class. That variable named as *SelectedRoute.*
There are methods for this class to save it *savePlan* and 2 methods for saving with different types text or voice. Also there are methods related with route by using our variable which points *Route* class element. These methods are *getRoutes* and *getShortestRoute*.

3.2 Place Class

Place class is most complicated and huge class among the others. Because there are a lot variables for this class and methods. These variables are *Name, Address, Time, Priority* and *ToDo.* This class is one of the main, core piece of the application.
Methods are related with crud actions among places such as *editPlace* and *deletePlace.* One of the functionality of application is generating route and places are shorted with a priority order so that we need methods like *setPriority* and *getPriority.* Also other get, set methods for identifier variables.

3.2 Route Class

Route class has less variables and methods from other classes however it has a lot of functionality through the application. Instead of identifier variables this time there are variables for calculating algorithms. These are *TotalDistance, NumberOfPlaces* and *EstimatedTime.* All of these variables are used at calculating algorithms.
Also methods are related with calculations too *calculateTime, getDistance, isFİnished* and *navigate.* These methods have a lot of other methods inside them and cost a lot of time because they are working on APIs and mathematical algorithms for routes.

4 Glossary

Plan: A detailed proposal for doing something

Place: A particular position in space

GPS: Global positioning system, accurate worldwide navigational system

Navigate: Plan and direct the route of a person from his/her current location

SQLite: SQLite is a relational database management system. It is a software library. This library has some features which are self-contained, serverless, zero-configuration, and transactional SQL database engine [1].

SQL (Structured Query Language): SQL is a standard language for relational database system. It is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. [1]

Database: Data storage system

UI: User Interface

5 References

[1] https://www.tutorialspoint.com

[2] http://www.cs.bilkent.edu.tr/CS491-2/current/OOSE-Documenting_UD.pdf

[3] http://dioscuri.sourceforge.net/docs/ODD_Dioscuri_KBNA_v1_1_en.pdf