Bilkent University

Department of Computer Engineering

Senior Design Project

Consigliere

Project High-Level Design Report

Selin Erdem, İrem Yüksel, Orhun Çağlayan, Furkan Küçükbay, Umut Mücahit Köksaldı

Supervisor: Assoc. Prof. Dr. Mehmet Koyutürk
Jury Members: Prof. Dr. Uğur Güdükbay
Prof. Dr. Cevdet Aykanat

Project Analysis Report
Dec 22, 2017

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Contents

1. Introduction

In the last few years, people's daily lives have changed in many different ways. Simple things became more complicated and we started to take more responsibilities and with more responsibilities, came higher expectations. Consequently, concepts such as making plans, using our time efficiently have gained more importance than ever before. Yet, it is a known fact that nowadays people struggle when it comes to planning a day and running errands. With the current technologies, it is possible to make a plan by entering the details such as time and place, and having reminder alerts for these tasks, but these applications do not help the user in terms of time efficiency and organization of their tasks.

Consigliere will be an iOS application that will work as a daily organizer and task manager. With Consigliere, we aim to help people use their time more efficiently and regain the time wasted on traffic. The user will simply need to enter whatever errands they have to run for that day and the application will provide an optimal plan for the user to complete these errands. This plan is designed by taking into account the roads the user will have to take to get to the locations of their tasks, and the traffic situation in the road. In addition, the application will estimate how much time the user will spend on an errand by analyzing the busyness of the location and the nature of the user's task. The application will also periodically check the traffic status of relevant roads and the crowd level of the locations in order to update the daily plan dynamically and send the user push notifications informing them of the opportunities to run their errands in a timely manner.

In this report, the overview of the architecture and design components of our system is provided. First the description, purpose and design goals of Consigliere are provided. Then, the quality and properties of existing systems are described. Afterwards, the system models of our system are included. The subsystem decomposition, architectural plans of subsystems, and hardware/software mapping of these components are illustrated. Design decisions such as persistent data management and boundary conditions are reported. Finally, the functions of subsystem services and their interactions are outlined.

1.1 Purpose of the System

The main purpose of Consigliere is to provide its users a practical and efficient scheduling tool that they can run their errands and plan their days in an optimized manner. For these purposes, Consigliere will offer number of features such as authorization, task managing, optimization of daily plans, route planning according to several crucial constraints such as traffic intensity. Furthermore, the application will optimize the way that users run their daily errands in the interest of saving time using different factors such as traffic intensity, road conditions and crowdedness. It will also offer several useful functionalities such as reminders.

1.2 Design Goals

1.2.1 Usability

The usability is an important design goal for Consigliere as the target users will use the app on a daily basis and it aims to make the users' lives easier. User-friendliness is a crucial factor for the app as the interactions to the system should be as simple as possible and user should not have difficulty to use the system. For this purpose, user interface must exhibit conceptual integrity and simplicity. Furthermore, novice users must be able to install the application and operate its major use cases with little or no training.

1.2.2 Supportability

The system should be adaptable to the future updates in external APIs and platforms that are used in application. Since the application depends on number of platforms such as Google Maps it is important for the system to be flexible to the changes in those platforms.

1.2.3 Reliability

Reliability is an important design goal for Consigliere as users will use it on a daily basis. They will need up-to-date information such as traffic data to be able to get an optimized and effective schedule. Hence, the traffic data used by the application must be up-to-date and accurate. Furthermore, the suggested routes must be accurate and route suggestions must consider temporary factors such as roads under maintenance and construction, or road closures. Our system also aims to avoid randomly scheduled daily plans and aims to address the preferences of the users.

1.2.4 Efficiency

Consigliere should be efficient in terms of computation and response times. The system should be able to complete the planning task under 10 seconds. The response time of the system should be less then 100 milliseconds.

1.2.5 Security

The application should ensure security of user's data & privacy. To address this goal, the application will be accessible only when user logs in to his or her account with his and her password. Furthermore, data that we access from other apps such as Google Maps or Google Calendar will be used only if the user gives permission. As users will share their personal data such as their home address, the system should also ensure secure data management and storage.

1.2.6 Scalability

The system should be scalable as the user can have arbitrary number of tasks per day. Therefore, our design goal is to make our application to support up to 15 tasks/stops per day.

1.2.7 Portability

The system should be able to be used in different software and hardware platforms.

1.3 Definitions, acronyms, and abbreviations

UI: User Interface
API: Application Programming Interface
Server: The part of the system which is responsible of logical operations, scheduling, and data management
Client: The part of the system that the user interacts

1.4 Overview

Consigliere is basically an application that enables users to plan and organize their errands in a time-efficient manner by suggesting dynamic daily plans. The applications main functionality will be mapping out a route that enables users to run their errands unerringly and efficiently. Main goal of our application is to help the users to finish his daily tasks in timely manner. It will also offer other several useful functionalities such as sending reminder notifications and saving parking spot to find the car easily.

What makes Consigliere different from classic task managers or map applications such as Google Maps or a regular daily planner is its incorporation of daily routine and task planning. Although most map applications provide route planning by selecting multiple points of interest, they fall short at optimizing the efficient route in terms of time and distance. Similarly, daily planners generally work just like agendas and only notifies the user about the upcoming events or tasks but they do not offer time management in terms of route optimization.

Consigliere will offer number of features such as task scheduling, optimization of daily plans, route planning according to several crucial constraints such as traffic intensity. Main challenge of this application is finding an optimized, relevant route that includes location of each errand that the user should run, which is basically an algorithmic and innovative approach to the travelling salesman problem [1]. Another goal of Consigliere providing an optimization for the way that users run their daily errands in the interest of saving time using different factors such as traffic intensity, road conditions and crowdedness.

2. Current Software Architecture

Current systems can be examined under two main functionalities: map applications and daily planner applications. Map applications generally gives user a time or distance optimized route between two or more points which user enters to the application. The map applications generally lack the optimization when it comes to calculate the route between more than two points. For example, Google Maps calculates the route between three nodes with given order (go to first point, then second point, then third point) even if going to the second point first shortens the distance.

Secondly, Daily planners lack the functionality of maps when it comes to planning the day. Generally, daily planner applications work as simple agendas and do not optimize the time or distance. Although there are several applications for calculating the route and planning the day separately, there are no application for planning the day on the map in a time-efficient manner.

3. Proposed Software Architecture

3.1 Overview

In this part, the subsystem decomposition of Consigliere is presented in detail.

- ✓ In 3.2, the partitioning of the system is displayed using UML diagrams together with the classes within tiers.
- ✓ In 3.3, hardware/software mapping of Consigliere is elaborated to show the allocation of the resources.
- ✓ In 3.4, the persistent data management is demonstrated and the database system is described.
- ✓ In 3.5, the access boundaries of users are defined and security issue in Consigliere is described.
- ✓ In 3.6, a general flow of the system is mentioned.
- ✓ In 3.7, the initialization, termination and failure conditions of Consigliere are provided.

3.2 Subsystem Decomposition

Our system is built on top of a Client-Server architecture model. The mobile application constitutes the client part of the system. The client requests services from the server to function and to respond the needs of the users. The server of the system is assigned the tasks of communicating with Google Maps API, making the necessary calculations to optimize the routes that the user will follow, creating errands and retrieving the data. The main reasoning behind selecting a Client-Server architecture model is to make our app battery-friendly and to be able to respond as many clients as possible without any significant performance penalty. Thus, scalability and high performance are given utmost importance. In short, the client will be focusing on taking the requests of the user and on responding to them while the server will be handling data retrieval and optimization tasks.

The Client-Server architecture model is further divided into a 3-Tier sub-model. The system has three layers: Presentation Tier in the Client side, Logic Tier and Data Tier in the Server Side. The Presentation Tier is the exact same of the Client and is responsible from interacting with the user. The Logic Tier in the Server side is the brain of the system as its name suggests and all the functionality resides within here. The data that are retrieved in the Data Tier are processed in Logic Tier. Finally, the Data Tier manages the database control.

.

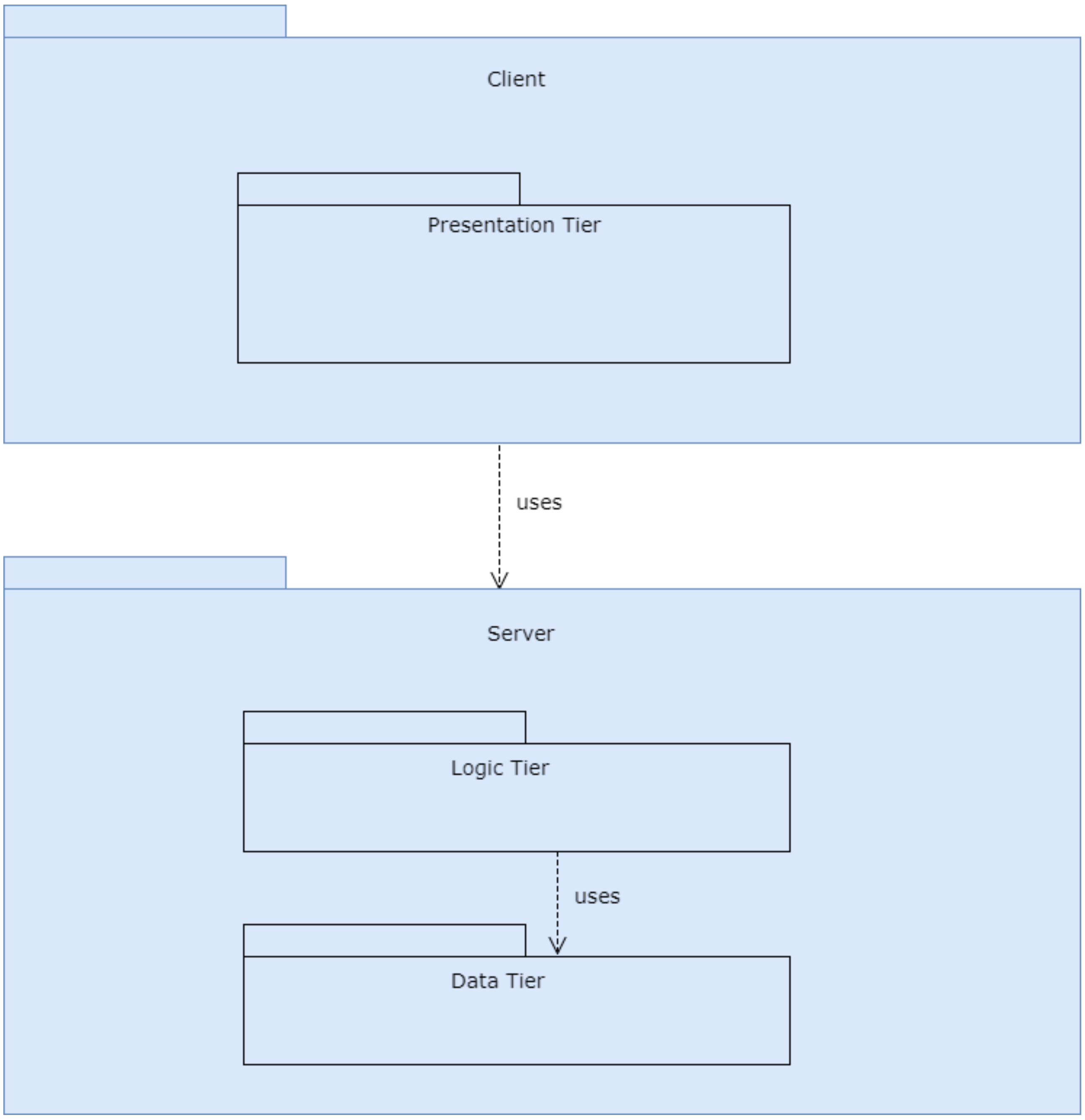

Figure 1. Subsystem Decompositon

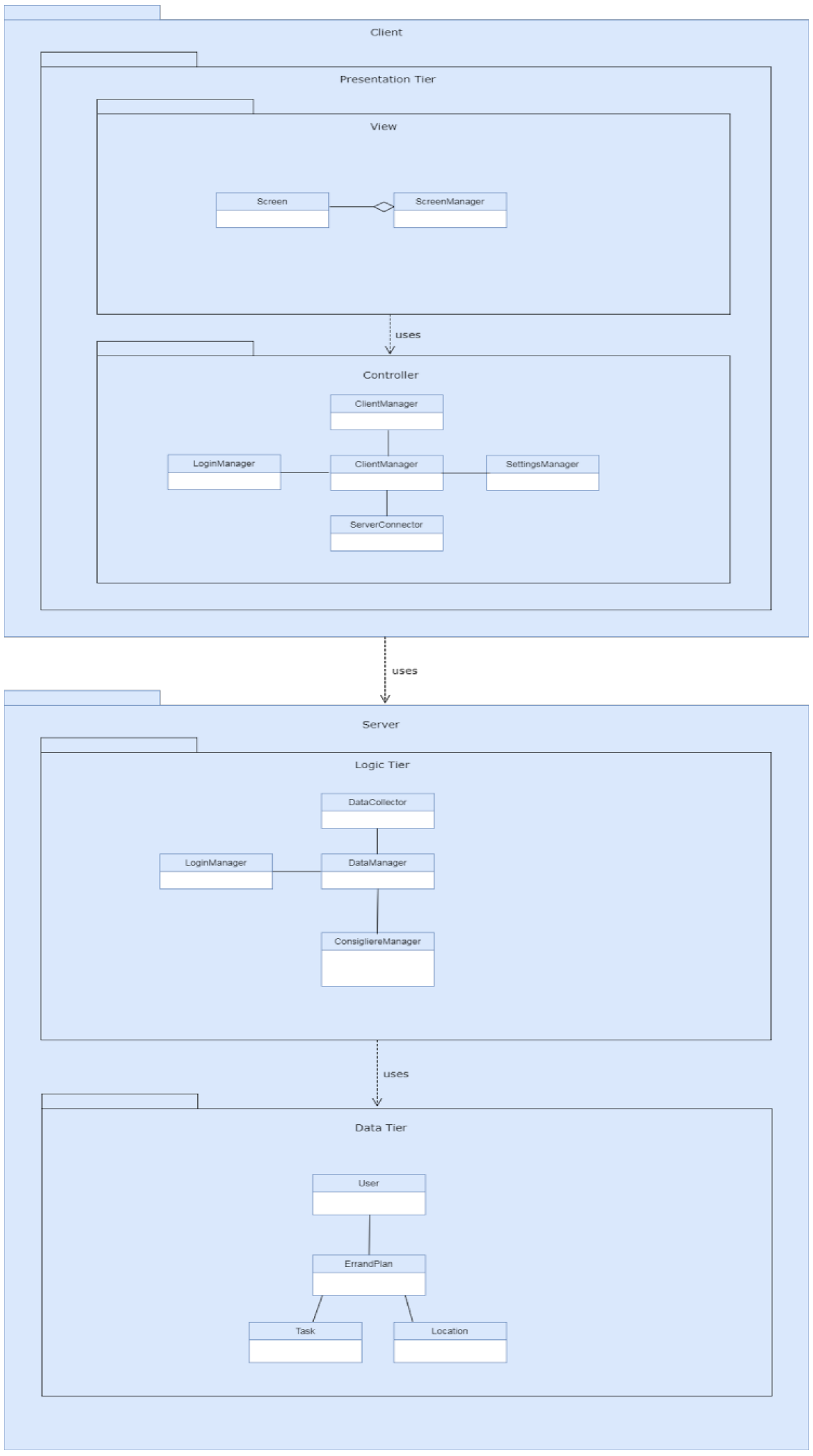

Figure 2. Subsystem Decomposition Detailed View

3.3 Hardware/Software Mapping

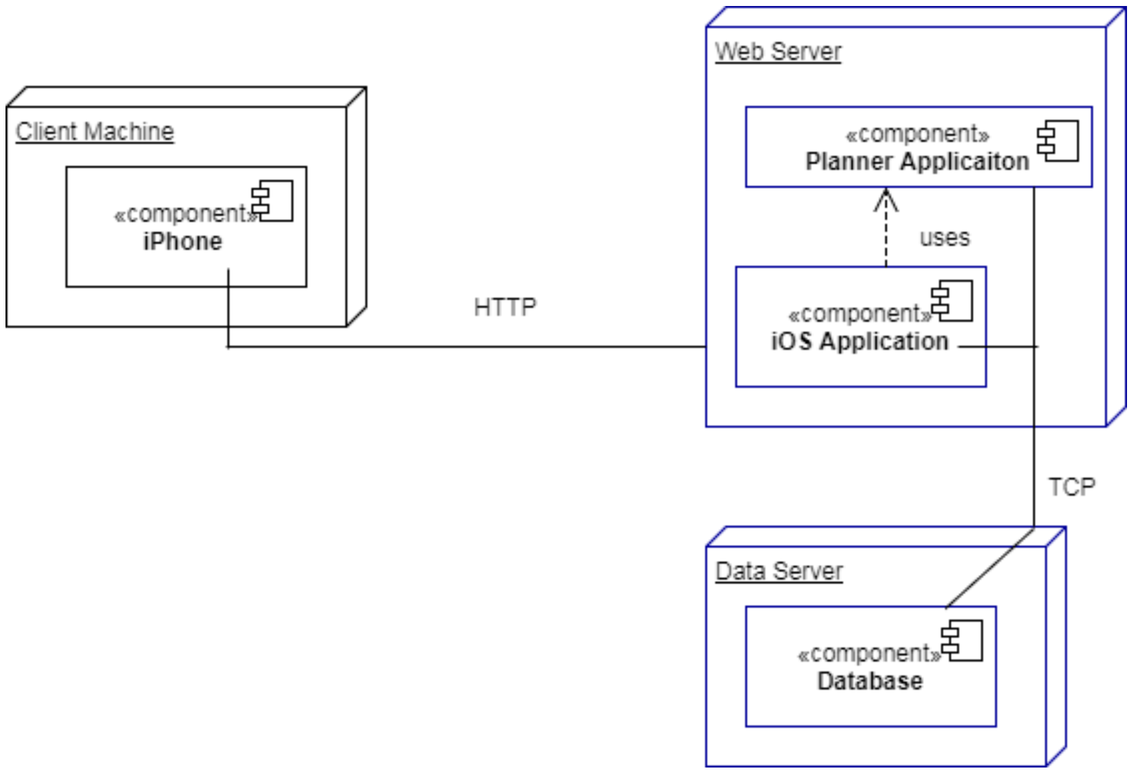

Figure 3. Component Diagram

The client application will reside in an iPhone. Consigliere will make use of phone screen to interact with the user and phone's location services to share with the server-side. Since the application data will be residing in the Data Server, the memory requirements of the application will be bearable. Internet connection will be necessary for the client machine to retrieve persistent and real-time data from the server side. The requests to the Web Server will be realized through HTTP.

3.4 Persistent data management

According to our plan, we have to store lots of data such as users, places, errands. Some of the data we will store will change constantly, users might change their errands, their locations and personal data all the time. As the data change over time, we might have to rearrange the activity plan and update the route. Moreover, we might have to response each changes very quickly. In that case, we cannot afford expensive database operations. Also, one of our aims is to collect data and specialize routes that we offer. Therefore, we need to use a database which is suitable for geographical data and more time efficient compared to other database systems. Hence, we decided to use PostGIS, an extension of PostgreSQL.

3.5 Access control and security

We will ask users to sign up and login to the program in order for them to use the application. Users will be allowed to change their errands, personal details and personal settings. However, they will not be allowed to change the overall map data.
For users to sign up, we will use an email/username/password sign up system. The user information will only be shown to the user itself. These data will be secured by using 3rd party security system. Users must login in order to access their user data, errands, and their route plans.
Apart from user request, we will not distribute irrelevant user information to 3rd parties. Also, application will not access or connect any other application without user's consent.

3.6 Global software control

We have an event driven system. The user logins to the system by using their username and password. Then the system checks if the username/password combination exists in the database. It the combination exists, the application gives the user permission to login to the app.
In the application, user can modify their personal information, errands, and check their route. When one of these are changed, app updates the user's profile.
When user enters a set of daily errands, app calculates an optimal route and shows it to the user. After each completed errand, user can label that event as 'finished' and app recalculates the route accordingly. User can request to look at the details of errands. For that request, app shows the details of requested errand.

User can request to see the details of points of interest on the map embedded into the application. In that case, the requested point of interest is shown on the map with its details. User can search for an address on the map. App then finds the searched address and shows it on the map.

3.7 Boundary conditions

3.7.1 Initialization

The user must have the application on their phone in order to use it. To use the application, user must create an account and use that account to log in to the application. After login, user can benefit from application. User cannot use the application if log in fails.
Additionally, application needs a regular internet connection in order to run, since it relies on live map data.

3.7.2 Termination

User can terminate the application by logging off. If he or she decides not to log out, application will keep the account information and keep the user logged in. Apart from logging off, user can also terminate the session by clearing the application data. If the application is not closed, it will run on the background.

3.7.3 Failure:

The application fails if there is no internet connection since live map data cannot be obtained. If the application is closed during the calculation of the routes, the routes cannot be saved.

4. Subsystem Services

This part of the report analyzes the subsystems of our system and describes the services they provide in detail.

4.1 Client Subsystem

The client corresponds to the mobile application of our system. The client is the presentation layer of our system. The user creates an account or logins to the system via the client. The client requests login access from the server. The client manages account of the user, the settings, the past/present/future errands and tasks. The user specifies the details for a new errand or a new task. The client is also responsible from presenting the data it collects from the server to the user, manage the operations of the user and send notifications when necessary.

Client subsystem includes Presentation Tier. Presentation Tier has View and Controller subsystems. View subsystem is responsible from all the user interface operations. Controller subsystem manages the interaction between the client and the server and it also controls the operations within the client. It collects the data from the mobile application and sends it to server. It also requests the required data such as daily plan from the server. Furthermore, Controller subsystem handles client functions such as account management, daily plan management, settings management.

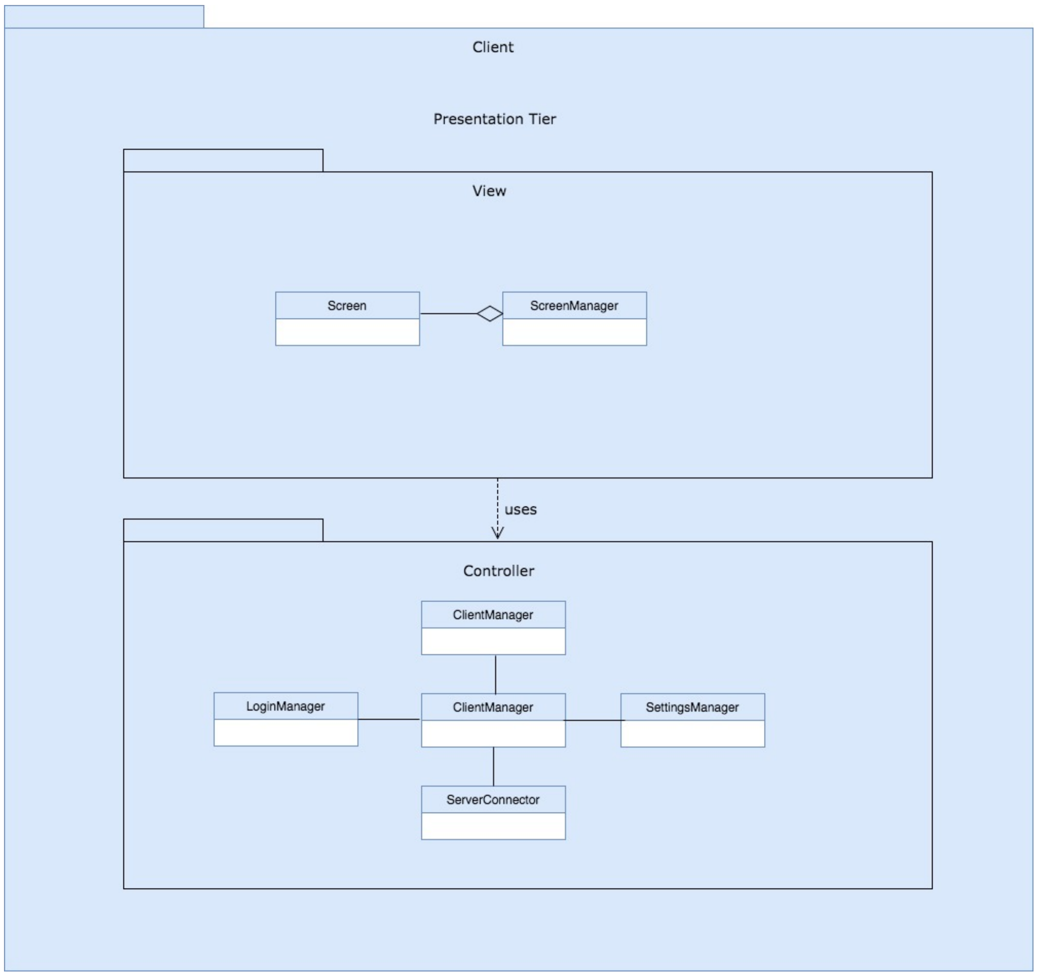

Figure 4. Detailed View of Client Subsystem

4.1.1 View Subsystem

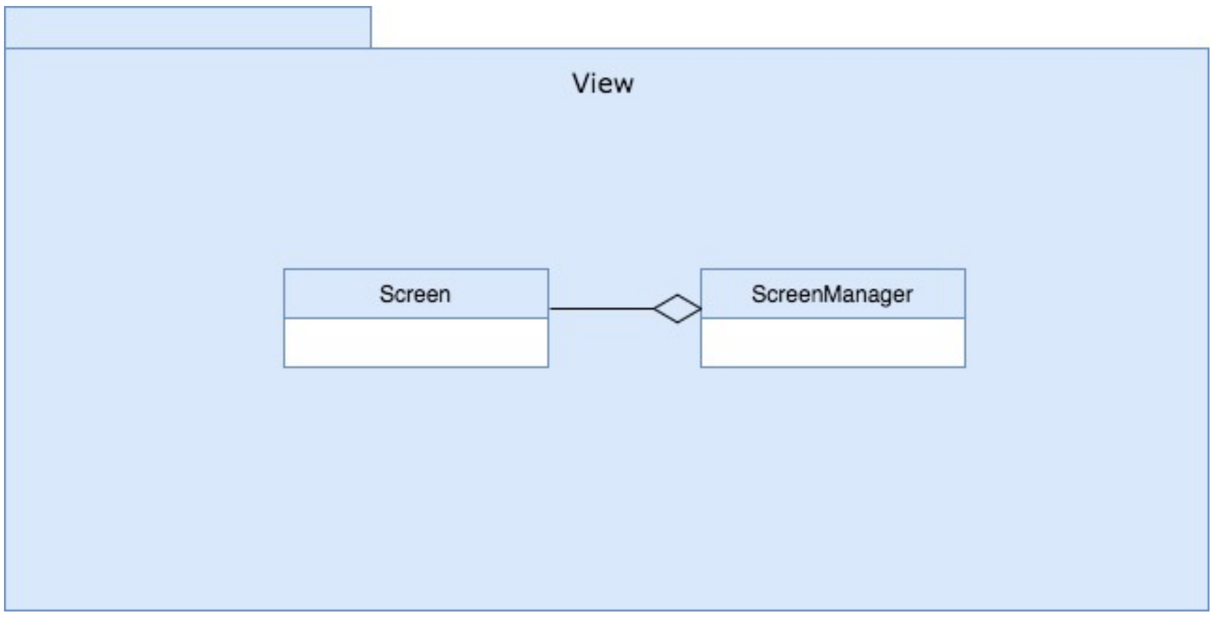

Figure 5. View Subsystem in Client

View subsystem handles all user interface related operations.

ScreenManager: Main manager class that arranges the tasks of other classes in the subsystem.

Screen: Class that is responsible from the presentation of all components in the screen of the user.

4.1.2 Controller Subsystem

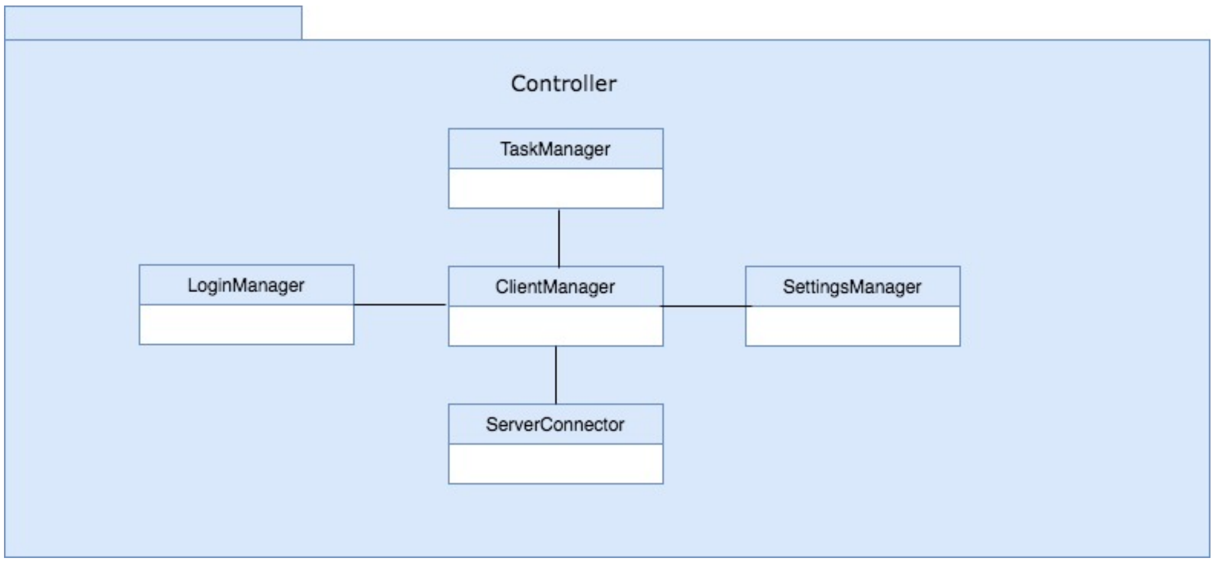

Figure 6. Controller Subsystem in client

Controller system manages the client operations and the interaction between the client and the server.

ClientManager: Main manager class that arranges all operations of other managers.

LoginManager: Class that collects the entered username/email/password inputs and manages the account login.

TaskManager: Class that collects the necessary information from the user about the tasks, sends the data to server, and collects the appropriate daily plan.

SettingsManager: Class that updates settings according to the requests of the user and notifies other components.

ServerConnector: Main class for handling interaction between the client and the server.

4.2 Server

4.2.1 Logic Tier

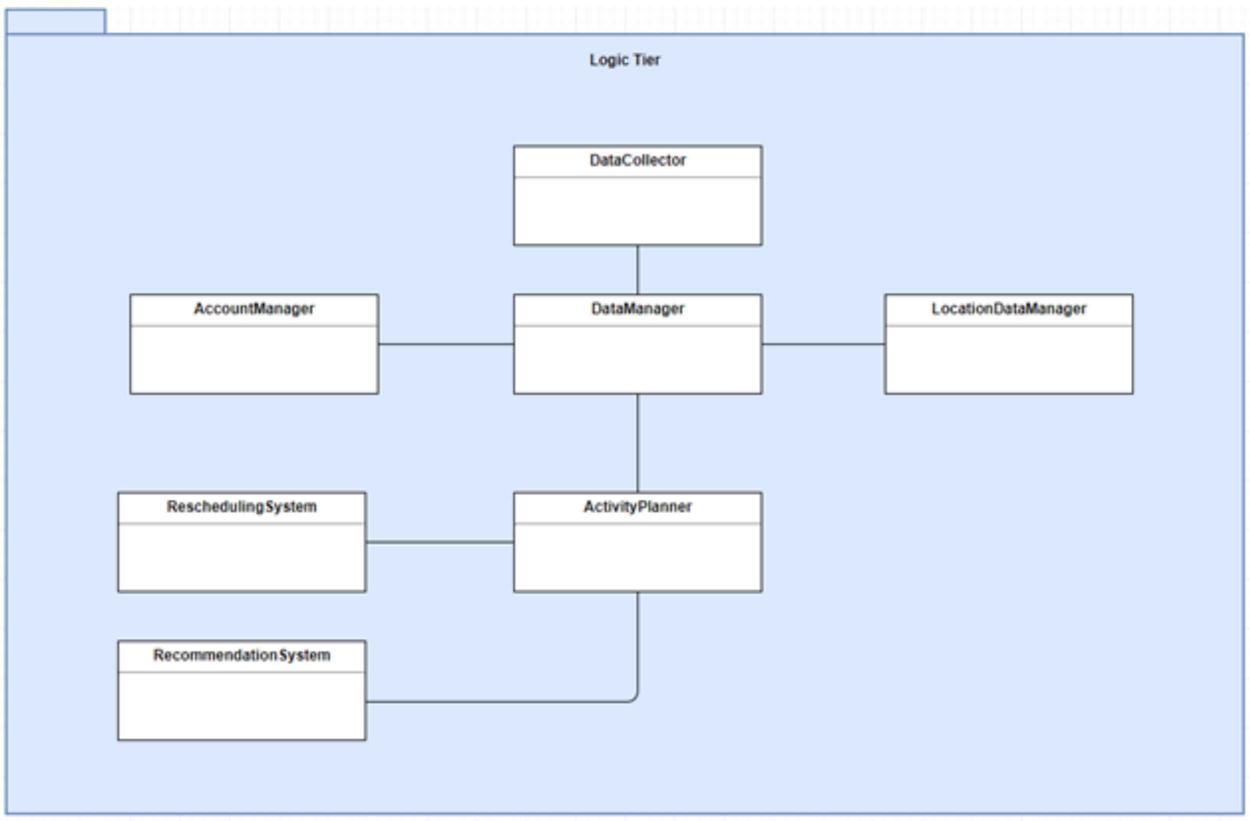

Figure 7. Logic Tier in Server

Logic Tier is responsible for the core operations that the application needs to do in order to provide the user with the desired data. Initially, the Logic Tier collects the information about the user's task from the client. Then it determines the locations of these tasks and checks the traffic info, as well as the crowd data of these locations. Finally, it constructs an ideal, tentative plan for the user and sends it back to the client.

DataCollector: This class is responsible for collecting the task and account information from the client and passing the information to the DataManager class. It will continuously check the active list of tasks, and the user's information about his/her schedule, and notify the server about whatever changes might occur.

DataManager: The DataManager class is responsible for obtaining the user related data from the DataCollector class, as well as managing the data stored in the Data Tier of the server. All data related to the user's tasks as well as user generated location data is managed and stored by this class.

AccountManager: The class that manages the account and authorization information of a user. Past tasks, on-going activities of a user, permanent schedule info related to a person's daily life as well as their credentials such as user name, password and e-mail address are managed by this class.

LocationDataManager: Interacts with the Google Maps API to collect information related to the location of the user's tasks such as traffic and crowd. It then passes the information to the DataManager class to be combined with the user's data.

ActivityPlanner: The core class that prepares the daily plan for the user according to the location and task data provided by the DataManager class. This class interacts with the ReschedulingSystem, RecommendationSystem and DataManager the create the most optimal daily plan for the user.

ReschedulingSystem: The class responsible for adjusting the current errand plan for the user should there be any changes related to crowd, traffic or user preference.

RecommendationSystem: This class is responsible for checking and prompting the user about the opportunities to complete some or all of their tasks in a timely manner, provided that the current conditions related to traffic and location are appropriate.

4.2.2. Data Tier

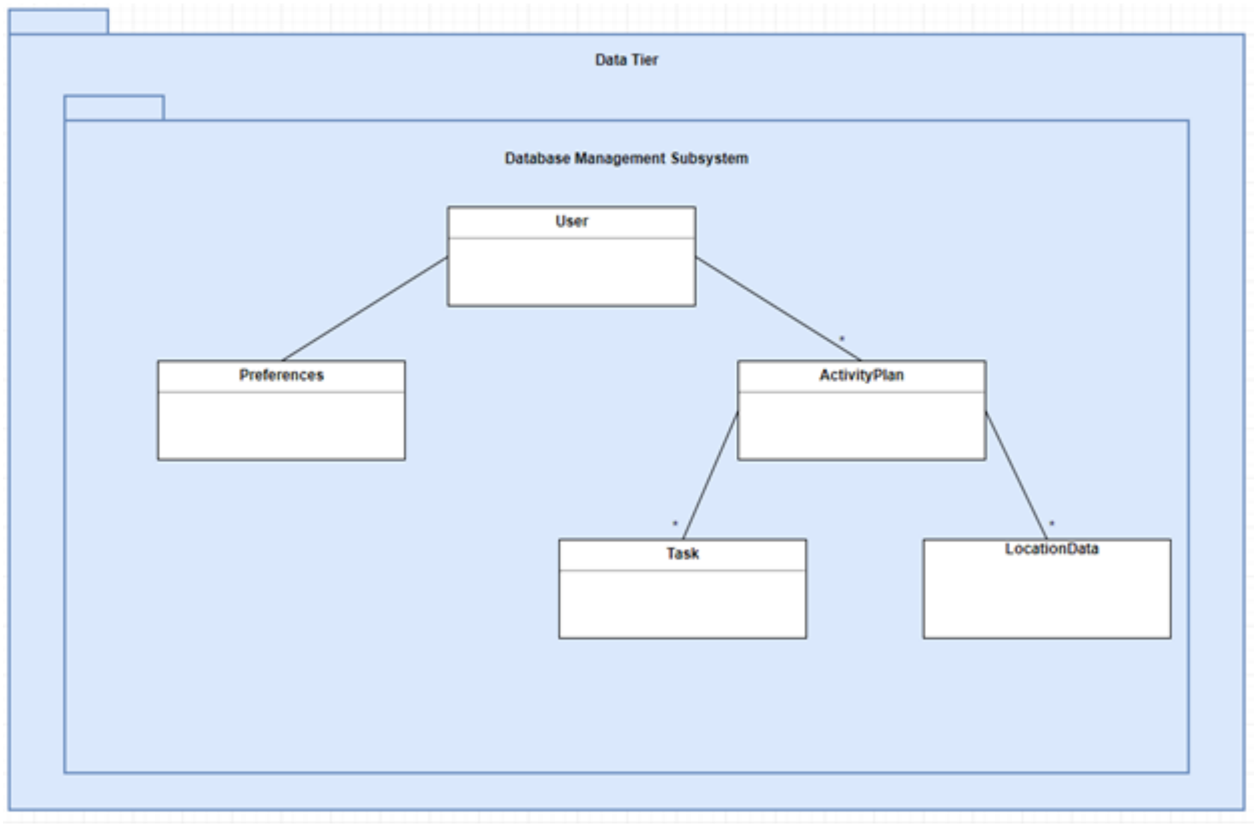

Figure 8. Data Tier in Server

The Data Tier of the server manages all data related to the application. The main component of this tier is the Database Management Subsystem, which manages the database located in the server which keeps track of the user's data as well as the data collected online.

User: Data class that represents the users of Consigliere. It includes credentials and the authorization information of the user such as e-mail, password, user name and account details.

Preferences: Includes information about the daily permanent schedule and the location preferences for various tasks of the user.

ActivityPlan: Data class that holds the current daily plan of the user, utilizing the information provided by the user in the form of task lists as well as the crowd and traffic information collected from the internet.

Task: Data class that represents a single task that the user has input into Consigliere. Holds the location information and the type of the task and the estimated duration of said task.

LocationData: Collects and holds information related to the transportation of the user from one task location to another. Includes information about the traffic situation on the roads between the tasks and the crowd levels of the task locations.

5. References

[1] "Traveling Salesman Problem", *Math.uwaterloo.ca*, 2017. [Online]. Available: http://www.math.uwaterloo.ca/tsp/. [Accessed: 15- Oct- 2017].