Product/Service Design

Introduction

**Title**

The title of our product offering is *Google Student*.

**Purpose**

The purpose of this design document is to provide an overview of the product we are building, along with some architectural considerations that will guide development. It will also serve as a continuously updated map of our current product and where we plan to go with it.

**Contents & Intended Users**

This document covers areas including the chosen design pattern, basic architectural design of the software system, and the needed modules for operation as well as the associated interfaces. It will also cover the main user profiles we designed this product for and the primary scenarios on how we expect our customers to use this product. Expected users of the document include the development team, testers, managers, and future developers seeking to extend or perform maintenance on the system.

**Desired Product**

The product we are creating is a smart device application that provides a variety of personalized information to UW Bothell students. The application will leverage Google Assistant platform technologies, and connect with information from both the Canvas LMS and from the smart parking system deployed on campus. The primary mode for interacting with the product will be voice, but some functions will also provide visual information and/or controls to the user.

**Customers/Users**

For this application, the customers are primarily college students at UW Bothell. Users may also include educators and other staff members associated with the campus. Initially we plan to launch this product to just UW Bothell, to test out the features and iterate through a few versions. Once we have a product that we feel is ready we will release it nationwide.

**Sponsoring Body**

Google Student is a project created by Google staff members using their ‘20% time’. The current iteration is to be used as a proof of concept to support a bid for greater funding from the organization. At some point in the future, the development team foresees the product being extended to many campuses across the US, and potentially resulting in revenue opportunities for Google.

Architecture

**Main Architectural Pattern**

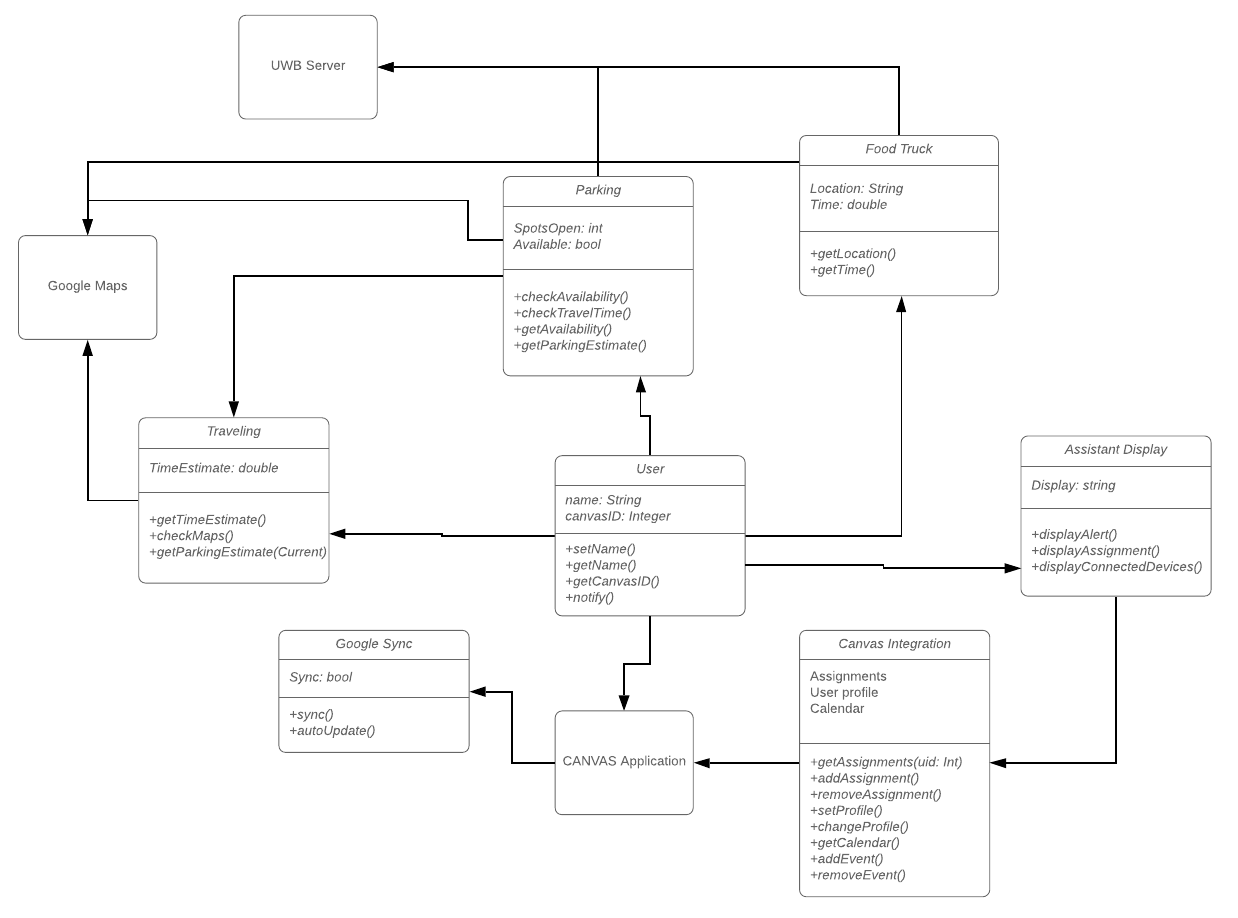
Table 1: Layered Architecture Pattern

|  |
| --- |
| Voice Assistant Application Phone Application |
| **Configuration Services** |
| Group Management Application Management  Identity Management |
| **Application Services** |
| Parking Availability Weekly Food Truck Schedule & Availability  Upcoming Events Reminders Degree Auditing Calls & Messaging |
| **Utility Services** |
| Logging in/out Authentication Interfacing User Storage |

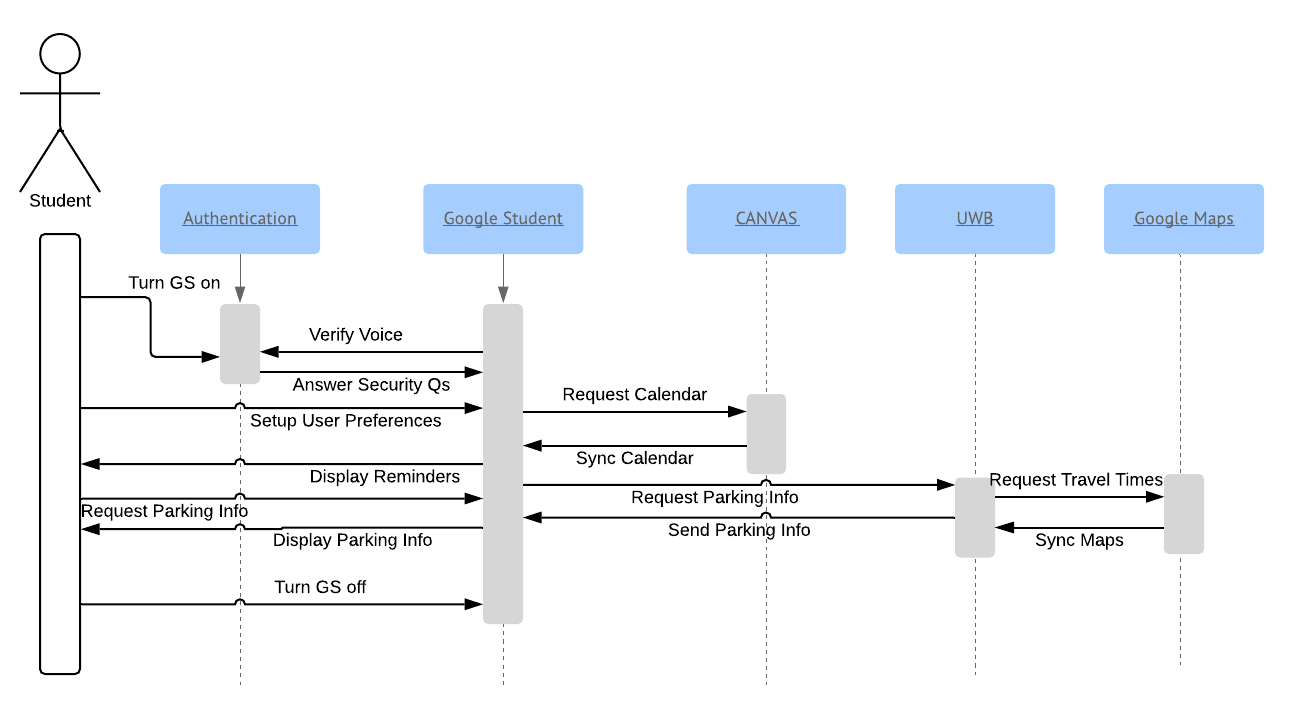
We chose the Layered Architecture pattern because it will allow us to be modular with the services that the product can provide by only needing to adjust a single layer as we add/remove services. This pattern organizes the system into separate layers, with related functionality associated with each layer. If we add more services that the application offers it doesn’t need to affect the utility or the configuration layers. This approach will also support incremental development.

**High Level Architecture**

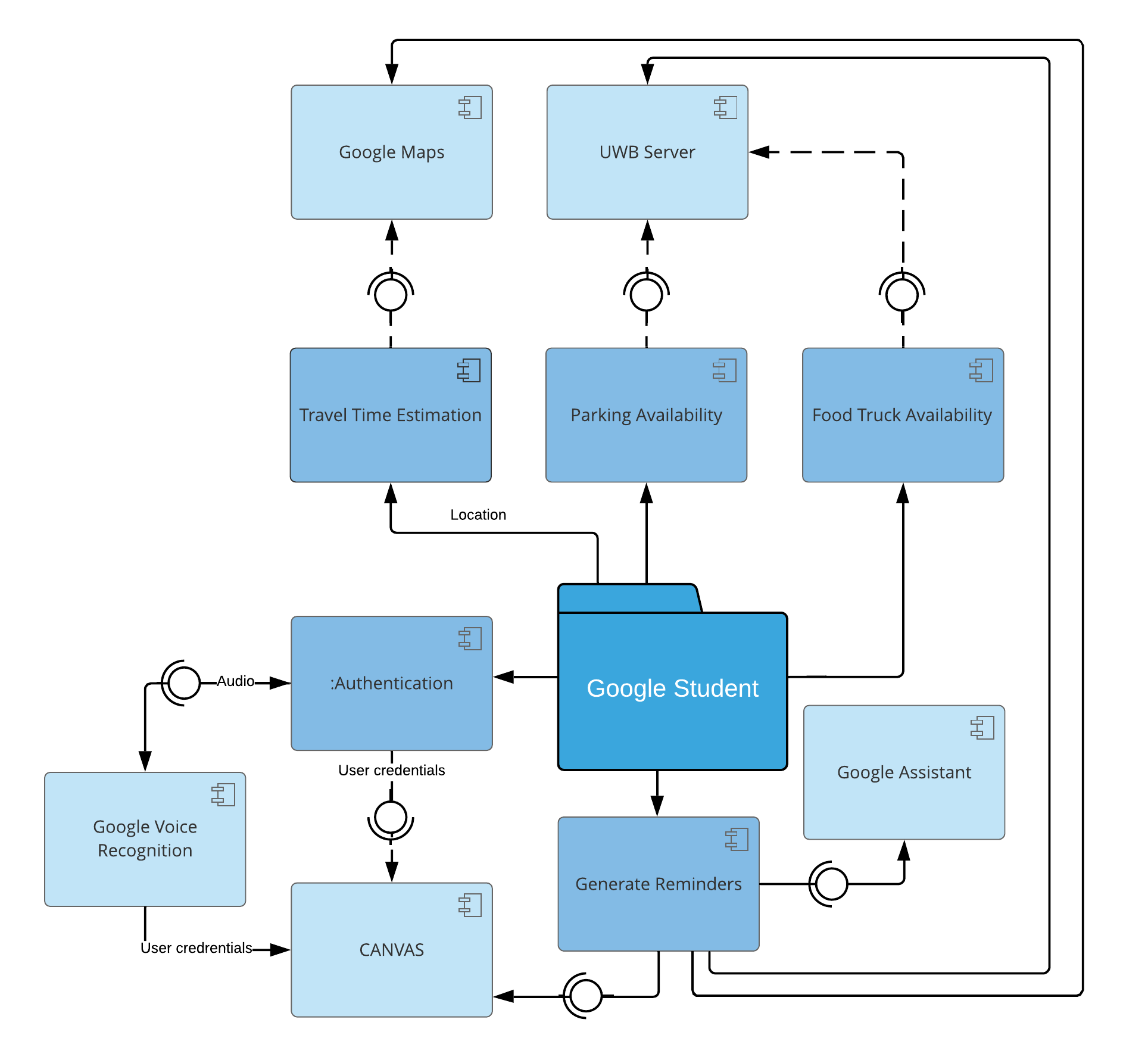
**Logical View - UML Class Diagram**



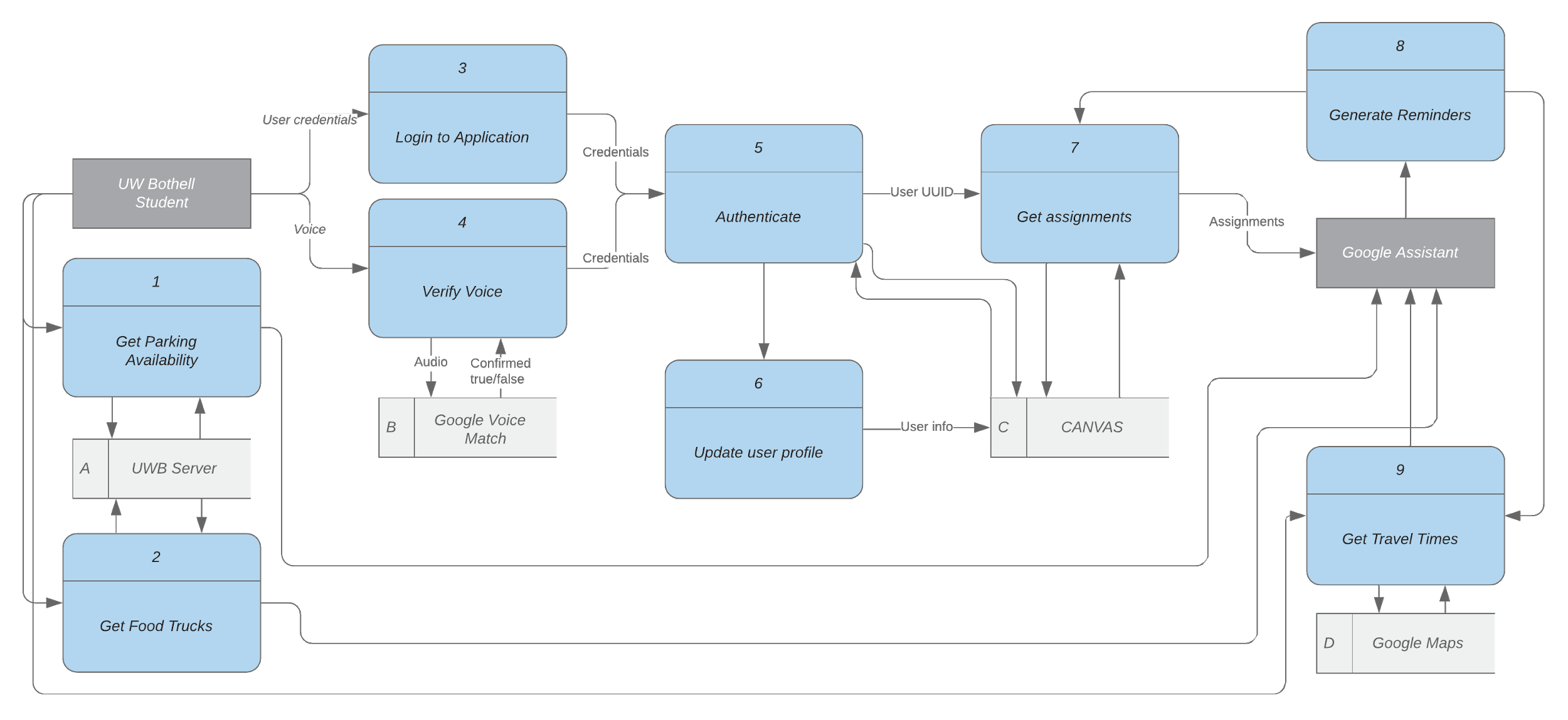
**Process View - Sequence Diagram**



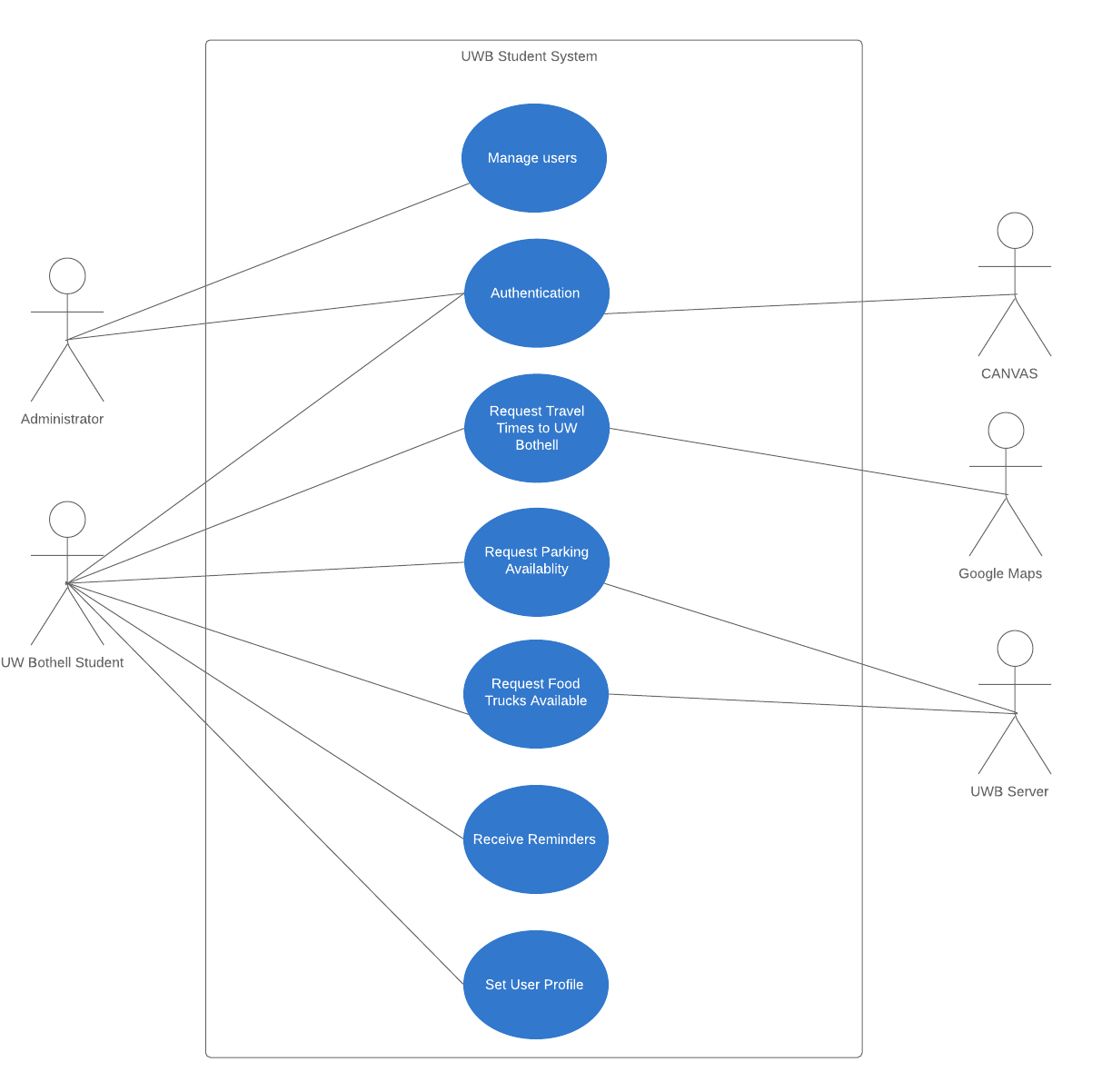
**Physical View - Software Component Diagram**



**Physical View - Data Flow Diagram**



**Scenario View - Use Case Diagram**



**Architectural Solution Reasoning**

The 4+1 Architectural View model was introduced by Phillipe Krutchen in 1995 as a model for providing different stakeholder groups with relevant information about the software or system being produced. “4” refers to the four essential views: Logical, Process, Physical, and Development; the “+1” describes the Scenario view.

The UML standard includes a number of diagrams capable of communicating the information described in each view. As such, development teams are responsible for choosing the most appropriate diagrams that best reflect the needs of each particular project.

The *logical view* ensures that the functionality desired by end users has made its way into the system design. As our team is using an object-oriented programming approach, we chose a Class Diagram. This diagram shows the objects that we intend to implement, their attributes and methods, and the interactions between objects. Alternatives to the class diagram include the object diagram, which conveys the relationships and values of objects at one point in time, and the state chart diagram, which attempts to describe the behavior of stateful systems.

The *process view* communicates to system integrators the tasks and processes carried out by the system, as well as how issues such as dependability and performance are addressed. For the process view, we chose a Sequence Diagram because we felt it would best capture the scenarios involved in our product, including the passing of data between systems at various times. Alternatives include the activity diagram, which works like a flowchart to represent control flow, and the collaboration diagram, which focuses more on the behavioral relationships between components.

The *development view* describes the modules, along with their dependencies and relationships, and is primarily for the use of developers. Because our system is relatively small, we elected to use a Component Diagram to communicate a more granular picture of the internals. The primary alternative representation for the development view is the package view, which places modules into groups, such as ‘Ordering’ and ‘Shipping’, particularly when the quantity of components makes it difficult to represent them effectively with a single chart.

The *physical view* examines the system at a hardware level, including server and network configuration, and is primarily used by system engineers and administrators. We opted for the Data Flow Diagram as our software product will run on Google’s internal cloud infrastructure (which may be serverless), abstracting away much of the value of a deployment diagram. This representation is in fact not part of UML, but rather, comes from DeMarco’s Structured Analysis methodology.

The *scenario view* demonstrates functionality by walking through various stories of how the system could be used. This view is important because it exemplifies requirements, illustrates the system for stakeholders, and serves as a starting point for validation testing. For our project, we chose the Use Case Diagram, as there appear to be few alternatives that satisfy this component of Krutchen’s model.

Module Design

**Major Module Reasoning**

Module 1: PARKING

|  |
| --- |
| **Purpose**   * Shows what parking spots are currently open, and estimates future parking lot fullness   **Provided Interfaces**   1. Parking Map: A map of all the general parking spots (excluding handicapped or EV charging spots) that is updated every 5 minutes the app is open 2. Parking Estimation: Guesses how full a parking lot will be on a future date/time   **Required Interfaces**   1. Parking Map: An up-to-date map on what parking spots are available and what full based on the information provided by the college parking system 2. Parking Estimation: A running heatmap of the parking map information that can be used to estimate how full a parking lot will be at any given time. 3. Event Reservation: Updated by the university showing which spots they have reserved for people visiting for the event that are normally available to the general public 4. Event Estimation: A special heatmap that overrides the general parking heat map for special events (career fairs, guest speakers, etc.)   **Constraints**   * Universities must interact with the program to reserve spaces * Universities must interact with the program to label events * Parking lots must provide metrics to the program to show which spaces are open or taken, and which are general public/EV/handicapped |

Module 2: TRAVELING

|  |
| --- |
| **Purpose**   * Help the user figure out how long their drive will be to get from their current destination to their school, as well as find parking   **Provided Interfaces**   1. Travel Time: This module will let the user input a time that they want to be at their school   **Required Interfaces**   1. Google Maps: Travel Time will connect with Google Maps and provide an interface for it, to allow the user to put in when they want to arrive 2. Parking Time: Adds on time to find parking in addition to just traveling to the school   **Constraints**   * Users must give consent to share their location |

Module 3: CANVAS INTEGRATION

|  |
| --- |
| **Purpose**   * This will interface with canvas and will include dates/time of events and assignments that the user has. It will also include alerts for assignments that start with a user preference and can be adjusted manually   **Provided Interfaces**   1. Canvas Assignments: Will automatically connect with canvas, and update the user on assignments when they are posted 2. Assignment/Event Alert: Alerts the user to an upcoming event or assignment based on their user preferences. This can be manually changed by the user by clicking on the event/assignment 3. Manual Entry/Updated: Allows the user to manually input assignments/events if they want to   **Required Interfaces**   1. Canvas: Requires the user to log into canvas, and give the program permission to access their profile 2. Calendar: Requires the user to have/use Google Calendar so Google Student can update their calendar. Also allows the user to manually input assignments/events 3. User Profile: One of the preferences in the profile the user can set is a preferred alert on assignments, and a preferred alert on events.   **Constraints**   * The user must allow the product to connect to Canvas * The user must use Google Calendar and allow the program to connect to it |

Module 4: GOOGLE SYNC

|  |
| --- |
| **Purpose**   * To keep Google Student updated to the current version   **Provided Interfaces**   1. Sync: The service will automatically update every 48 hours when connected to the internet   **Required Interfaces**   1. Sync: This will be a backend module that the user will not need to see, but is taken into account to keep the application updated |

Module 5: ASSISTANT DISPLAY

|  |
| --- |
| **Purpose**   * To display the users schedule and events on any connected Google assistant device   **Provided Interfaces**   1. Assistant Sync: Looks for connected devices, and displays the users information to the device   **Required Interfaces**   1. Connected Devices: A list of devices that the user has connected to 2. Student Display: Displays and alerts the user to upcoming events/assignments   **Constraints**   * Only displays on Google devices the user has connected to with their phone |

Module 6: FOOD TRUCK TRACKING

|  |
| --- |
| **Purpose**   * This will help the user know which food trucks are available and their location, and the times they are on campus everyday.   **Provided Interfaces**   1. Food trucks availability: Looks for food trucks available on Campus and alerts the user 2. Food trucks mapping: Tracks exact location of food trucks once they arrive on campus   **Required Interfaces**   1. Google Maps: connected to maps to track exact location of food trucks on campus   **Constraints**   * Food truck owner/driver must be willing to give information about their location |