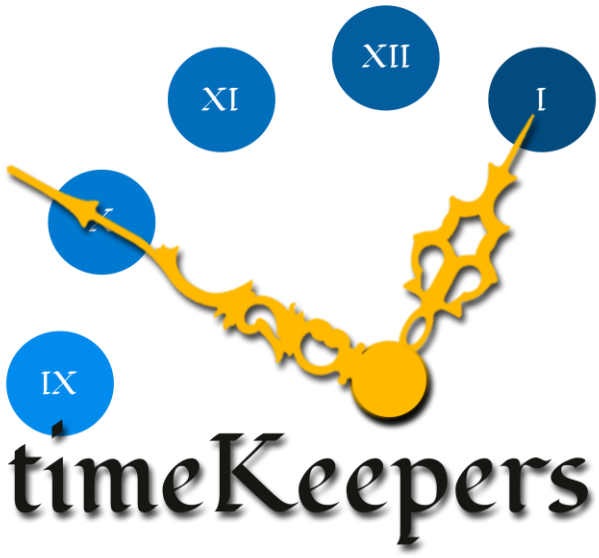
Department of Computer Science and Engineering
The University of Texas at Arlington



Team: TimeKeepers

Project: Volunteer Tracking System

Team Members:

Dineth Hettiarachchi
Damber Khadka
Devkishen Sisodia
Samir Shrestha
Tasneem Devani

Table of Contents

Document Revision History

Revision Number	Revision Date	Description	Rationale
0.1	12/03/14	Official First Draft	First draft complete
0.2	12/07/14	Revised ADS Draft	Major revision to the Architecture of the system; therefore, updated all the sections affected by the change
0.3	12/08/14	Peer-Reviewed Draft	Incorporated comments and suggestions from Team Ground Control
1.0	12/08/14	Review Ready	Validated the consistency and the formatting of the document; the Draft is ready for review
2.0	01/21/15	ADS Baseline	Added analysis to the Producer-Consumer Relationships Table based on feedback received from draft review

List of Figures

List of Tables

1. Introduction

This Architectural Design Specification (ADS) document describes how the TimeKeepers will design the Maverick Volunteer Tracking System and the Android app. This document will also describe the system's meta-architecture and describe the details of each architecture layer and subsystem. This introduction will provide a concept of the website and the Android app, proceed to cover the scope, and provide a high-level overview and interaction between each layer and their subsystems.

1.1 Product Concept

In the College of Engineering, there is currently an organization, Maverick Volunteers, which allow the various members of the Board of Advisors to volunteer and participate in different service opportunities. An administrator manually maintains the current system. The Maverick Volunteer Tracking System seeks to solve this problem. The main purpose of this project is to provide a system to the volunteers in the College of Engineering Board of Advisors to input, track and analyze their volunteer activities. To keep track of different activities or upcoming opportunities, the system will send a notification to all the volunteers.

In addition to the website, an Android based mobile app will be developed. The purpose of this app is to provide an ease of access to the Volunteer Tracking System. The Android app will require Internet access to input data into the system. The app will allow the volunteers to access the same functionality as the website. However, the functionality of the Admin and Facilitators is limited. To access their unique functionality, an Admin or a Facilitator would need to directly access the website. The functionality on the app is limited as the TimeKeepers will be primarily focused on the website.

In the future, the Volunteer Tracking System may be open to students, faculty and staff from the College of Engineering or other departments around the campus.

1.2 Project Scope

The TimeKeepers are designing a website and an Android application that will provide an efficient and interactive way for the Maverick Volunteers to log their volunteer hours as well as to keep track of their volunteer activities. The volunteers will be able to keep themselves updated about the upcoming volunteer opportunities and periodically view their progress report. The system will also provide a means to the facilitators to track volunteer participation and use it as a means to determine strategy for increasing volunteer participation.

Based on our current analysis the system will have three levels of users, which include admin, facilitator and volunteers. The facilitators will be able to add new events to inform the volunteers and the volunteers will be able to log their time, signup for an event and view their progress. The admin will be able to manage all the users as well as make changes to the system content.

The Volunteer Tracking System can be easily accessed under the uta.edu/engineering/ webpage or through any phone with android version 4.1.2 or above. The app will be available to download from the Google Play Store. Once the app is downloaded onto the phone, the user will start it for the first time. This will require them to enter their Email and password used to create the account. Once they have been validated, the internal database syncs and receives the data from the external database to display the necessary information to the user. The various users of the system will only be able to access the functionality corresponding to a Volunteer. The Admin and Facilitator will need to access the website in order to access their unique functionality. The Volunteer Tracking System will aim to fulfill all of the requirements listed below in Table 1.3.

The TimeKeepers aim to complete a working prototype of the website by the end of January. The project will be developed within the max budget of $800. The main source of cost for this project, $25, comes from publishing the app in the Google Play Store.

The different subsystems later in the document will explain the process the app and the website follow to connect to the databases and retrieve information.

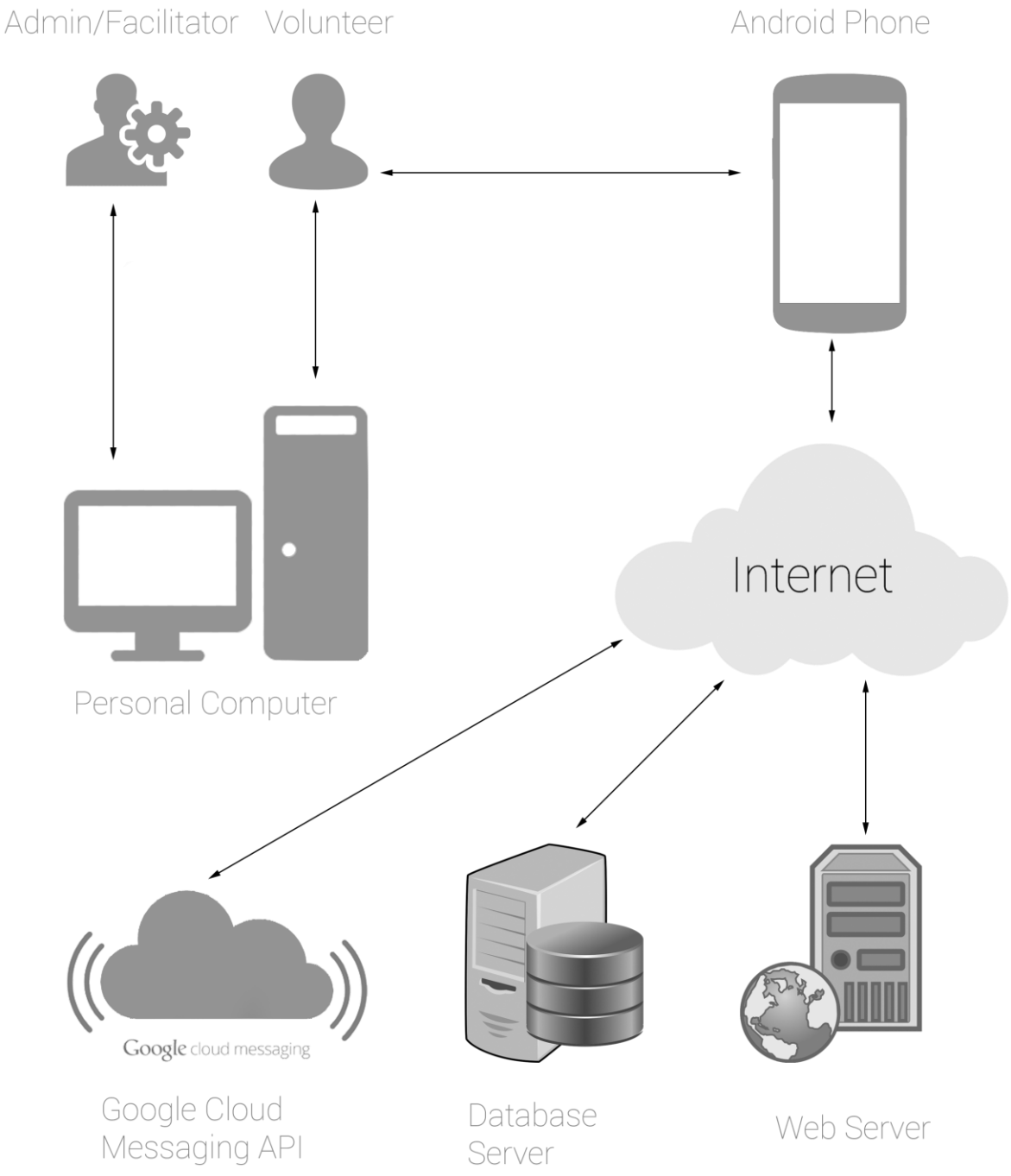


Figure 1.1 High Level System Diagram

1.3 Key Requirements

Requirement No.	Name	Description
3.1	Input Volunteer Hours	The Volunteer Tracking System shall allow a user to input the hours volunteered. To input the hours, the volunteers shall be able to select the name of the opportunity from a dropdown associated with a category and enter the number of hours they volunteered along with comments. If a volunteer does not commit to an opportunity, but still volunteers at that opportunity, the system shall allow them to input the hours they volunteered.
3.2	Notify Admin	The Volunteer Tracking System shall also notify the admin and the opportunity category facilitator when members input their time volunteered. This notification will be system generated. The admin and the facilitator will be able to see the notification upon logging in to the system.
3.3	Input Volunteer Hours on Behalf of User	Upon the request of the volunteer, facilitators must be able to input the volunteer hours on behalf of the volunteer. The facilitators shall be able to see a list of all members and an option to input their volunteer hours. The facilitator will have access to input the volunteer hours of all members without any time limitations or constraints.
3.4	Add Volunteer Opportunities	The Volunteer Tracking System shall allow facilitators to input the new or upcoming volunteer opportunities. An opportunity may include a title, description, date and time, location and images.
3.5	Delete Volunteer Opportunities	The Volunteer Tracking System shall allow facilitators to delete volunteer opportunities previously entered into the System. If volunteers have committed to an opportunity and it is cancelled, the system will notify all volunteers through Email.
3.6	Sign Up for Volunteer Opportunities	The volunteers shall be able to see the details of an opportunity such as the date, time, and location and have an option to sign up for an opportunity to indicate they will be volunteering at that opportunity.
3.7	Cancel Commitment	The volunteers shall be able to cancel a commitment they previously made. If volunteers previously signed up for an opportunity, the system shall allow them to cancel their commitment to indicate they will no longer be volunteering at that opportunity.

3.8	Notify Volunteer	The Volunteer Tracking System shall notify the volunteer and the opportunity facilitator upon the volunteer's acceptance/commitment or cancellation of an opportunity. This notification will be system generated. The volunteer and the facilitator will be able to see this notification on their home page.
3.9	Track Progress	The Volunteer Tracking System shall allow users to track progress of their volunteer activities and the status of different service levels. Service levels are different levels that volunteers can achieve based on the total number of hours. The levels are divided as follows: 30, 60, 90, 150, and 150+.
3.10	Generate Reports	The Volunteer Tracking System shall generate progress reports for each volunteer upon their request. The progress report should include details such as the categories/types of opportunities volunteered in, and the total number of hours volunteered.
3.14	Customize Preferences	The Volunteer Tracking System shall allow volunteers to customize their preferences. Preferences include setting the date of availability along with level of interest in different opportunity categories.
3.15	Login	The Volunteer Tracking System shall allow users to login with their Email and password. When a user logs in to the system for the first time, the system shall allow them to enter their Email for validation. When the Email is validated, the system shall ask the user to establish their password. When a user logs in to the system again, they will be required to enter their Email and Password for validation.
3.16	Logout	The Volunteer Tracking System shall allow volunteers to logout of the system. When the user is logged out, the system shall redirect to the login page.
3.18	Register Volunteers	The Volunteer Tracking System shall allow Admin to register volunteers and allow access into the system.

3.22	Ease of Use	The Volunteer Tracking System shall provide a user-friendly interface. The system shall also limit the number of clicks to allow a user to reach their desired page easily.
3.23	Android Application	The Volunteer Tracking System shall be available in the form of an Android Application. The Application will be available in the Google Play Store to download for free.
6.2	Password Encryption	All the user passwords shall be encrypted in the MySQL database.

Table 1.3 Key Requirements

2. Meta Architecture

This section describes the various guiding principles that were used to design this system's architecture. These guiding principles serve as the foundation upon which the rest of the document is based upon. This section also explains the architectural vision, assumptions, and tradeoffs.

2.1 Architectural Vision

TimeKeepers have designed the architecture in a way that all of the layers and the components associated with the layers are simple and extensible. The architecture is designed in this particular way so that in the future, it leads to an ease of development and modification.

The Volunteer Tracking System will have four layers with several subsystems interacting with each other within and between layers. The four main layers are: the Presentation Layer, the Application Layer, Service Layer and the Data Storage Layer. The Presentation Layer interacts with the user, obtains the input and displays the requested information. The Application Layer will gather that data and process it appropriately so that it is retrieved and formatted correctly. The Service Layer allows Web and Android apps access to the Volunteer Tracking System. The Data Storage Layer stores the Internal (Android) and External (Remote) databases associated with the website and the app, respectively.

2.2 Guiding Principles

Before designing the architecture, a few guiding principles will serve as the foundation upon which the Volunteer Tracking System will be designed. These principles allow the team to have a better understanding of the various aspects to consider while designing and implementing the system.

2.2.1 **Ease of Use:** The primary users of the Volunteer Tracking System will be the College of Engineering Board of Advisors. The website and the app will be intuitive and easy to use. To traverse from page to page, the system will keep in mind the number of clicks that it takes for a user to get from one page to another.

2.2.2 **Extensibility:** The Volunteer Tracking System will be designed in a manner that is easy to maintain and extend. It will be easily expandable as there is also a possibility that the system can be open to students and faculty of UTA in the future. Also, because of this expansion, the System may need an iOS companion app, in addition to the Android app. Therefore, the Volunteer Tracking System is designed in a way that it can be easily extended.

2.2.3. Modularity: The Volunteer Tracking System will be designed in a way that different layers and the subsystems will be independent of each other to allow for ease of future modifications. The system will be flexible to include future modification such as addition of different user types.

2.2.4. Performance: The Volunteer Tracking System will be designed for rapid performance. The user experience will be smooth in a sense that when they navigate to different pages in the system, the response and performance should be quick. Each layer will be designed in a manner that allows it to function in the quickest way possible. Other processing actions that could hinder the performance such as queries are delegated to the Service Layer that separates out the different components involved and reduces the overall time required for the processing.

2.2.5 Reliability: The Volunteer Tracking System will be designed to be reliable. The system will aim to allow for accuracy of data between the app and the website. If the number of hours is updated by a volunteer on the website, the system should be reliable so that app is able to sync with the external database and obtain that information in the app to display to the user.

2.3 Assumptions

The following are TimeKeeper's assumption for the Volunteer Tracking System's design.

2.3.1 Internet Access: The users using the website will have Internet access to view and use the system's functionality. The users of the Android app will have Internet access to update any information in the database. For viewing purposes, it is not necessary to have the Internet access on the Android app.

2.3.2 Volunteer Integrity: The users of the system will be entering various inputs such as the number of hours they volunteered. The TimeKeepers are assuming that Volunteers will enter their hours accurately without the need for any verification procedure of the number of hours entered.

2.3.3. Android Device: The users of the Volunteer Tracking System will have an Android device to use the application on a phone. The Android device must be capable of running Android software version 4.1.2 or higher.

2.4 Tradeoffs

In this section, tradeoffs will be discussed in detail. These tradeoffs reflect the guiding principles and the cost associated with the different tradeoffs.

2.4.1 Ease of Use vs. Website Organization: Ease of use is one of our main guiding principles. The users of this system have requested a maximum of three clicks to navigate to a

particular page therefore the website will need to be reorganized in the same manner. The website will be organized in a way that allows for this ease of use for the users.

2.4.2. External vs. Internal Database: The Volunteer Tracking System includes both a website and an Android app. This causes us to consider two databases. If the website was hosted on a choice of our domain, it would have reduced some of the synchronization handling. This website is being hosted under uta.edu/engineering, therefore the internal database cannot directly access the external database. The Android app will need an internal database to store all the local information and access the data upon the user's request. When the users do not have the internet access or the connection is limited on their mobile devices, they will still be able to view and navigate to different areas using the internal database.

2.4.3. Performance vs. Security: The TimeKeepers will design the system so that it is fully secure and take measures to prevent any malicious activity from occurring. Performance is crucial to the system as it is one of our guiding principles and top priorities when designing the system architecture. The security will need to be comprised in order for the performance to improve. However, for the security only certain parts of the database need to be secure, for example the password of a user's account.

3. Layer Definitions

This section describes the high-level layer structure of the system. It also defines in brief what each layer does and what subsystems is it composed of. The Maverick Volunteer Tracking System will consist of four layers: Presentation Layer, Application Layer, Service Layer, and Data Storage Layer. The figure below shows the layers that the system will be composed of.

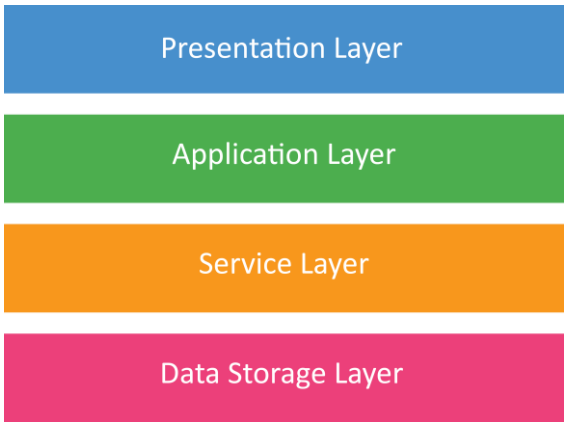


Fig: 3.1 High Level System Layer Structure.

3.1 Presentation Layer

The Presentation Layer consists of the Web App GUI, Android App GUI. This layer is responsible for gathering input from the user and displaying processed information back to the user. The Web App GUI consists of all the website GUI's that will allow the user to interact with the system through their computer. Similarly, the Android App GUI consists of all the GUI's that will allow the user to interact with the system through their Android smart phone. The layer below the presentation layer is Application Layer. The Presentation Layer will pass on the raw data collected from the user as input to the Application Layer. Depending on the task, the Application Layer will pass the processed data back to the Presentation Layer, which will then be able to display the information back to the user either through the Web GUI or the Android GUI. Overall, the guideline used to define this layer was that there needed to be a layer to handle all the user interaction and information display.

3.2 Application Layer

The Application Layer is the next layer in our hierarchy of layers. This layer is composed of Android Controller, Web Controller, and Client Transport subsystems. The Application Layer communicates with

the presentation layer to get the user input and events associated with user actions. Depending on the nature of the input, this layer either needs to request information from the database or store data in the database through the Service Layer. The Android Controller will be responsible for handling all the events generated by the user through the Android GUI whereas the Web Controller will be responsible for handling all the events generated by the user through the Web GUI. The Controller will then pass the data collected to the Service Layer through the Client Transport subsystem. The Client Transport acts like a bridge between the Application Layer and the Service Layer.

Overall the guideline used to define this layer was that there needed to be a layer that separates the GUI from the background processing and also acts as the communication bridge from the client side.

3.3 Service Layer

The Service Layer is the next layer in our hierarchy of layers. The Service Layer is composed of the Server Transport, the VTS API, VTS OAuth subsystems. This layer is responsible for doing all the server side processing and direct database access. The Application Layer transfers the data to the server layer through the server transport subsystem which then uses the VTS OAuth subsystem to verify the session before doing any processing. After the session has been verified the VTS API which houses all the protocols for data input and retrieval will access the database through the Server Transport. The Server Transport acts like the main communication line between the client and the server side to either pass data to the VTS API and VTS OAuth or to send the JSON response back to the Application Layer.

The guideline used to define this layer was that the client side processing needed to be separated from the server side processing. This will facilitate a highly cohesive system with low coupling. This will also facilitate component testing as the client side and server side code can be tested independently. Since the Service Layer can only access the databases, the Application Layer and the Presentation Layer will be greatly simplified.

3.4 Data Storage Layer

The next layer in our layer hierarchy is the Data Storage Layer. This layer consists of the Android and remote databases. The Application Layer communicates with the Data Storage Layer through the Service Layer.

The web application will only communicate with the remote database that will be implemented as a relational MySQL database. However, the Android application will communicate with both the Android database that will be implemented as relational SQLite database and the remote MySQL database. The presence of an Android database will allow the user to view their past activity.

Overall the guideline used to define this layer was that there needed to be a layer that allows separation of data from the program itself and also allows for easy access to the data. Hence, the only responsibility of this layer is to store data efficiently and allow secure access to the data.

4. Inter-Subsystem Data Flow

This section illustrates the high-level system architecture of the Volunteer Tracking System. That is how the user interacts with the system, how the system will be built, the different layers of the systems, containing sub systems of each layers and the data flow between different sub systems. This section also explains the different types of data elements that flow in/out of the subsystems and their behavior. Lastly, the section illustrates the relationship between data producers and their respective consumers.

4.1 Data Flow Diagram

Volunteer Tracking System consists of two components: Web App and the Android App. Both of the above apps are access via its Graphical User Interface (GUI). Therefore, the system consists of dedicated GUI subsystem for each of the component in the Presentation Layer. Each GUI has a corresponding Controller that is responsible for providing necessary information to display by the GUI subsystems as well as in sending data to the system. This data transfer is handled by the Client Transport subsystem. Therefore, Android Controller, Web Controller and Client Transport subsystems lay on the Application Layer. Service Layer is responsible for fetching data from the Remote Database and make them available for the Apps via an API. Therefore, VTS API (Volunteer Tracking Application Program Interface) subsystem is the key subsystem in this layer. It allows external access to the system via API calls. VTS OAuth subsystem is responsible for user validation. That is user login, signing up or even the API calls that are user specific have to go through the VTS OAuth subsystem. The subsystems in the Service Layer use Server Transport subsystem to communicate with the other layers. Data Storage consists of two subsystems: Android Database and Remote Database. Android Database lies within the Android App while the Remote Database is stored in a different server from the server where the API is hosted. Therefore, Android Database is directly accessed via the Android GUI subsystem. Remote Database can be accessed by both Android and Web App. Google Cloud Messaging Service is used as an external service to notify of the Remote Database updates to the registered Android users. The diagram also illustrates the need for Internet access for relevant communication.

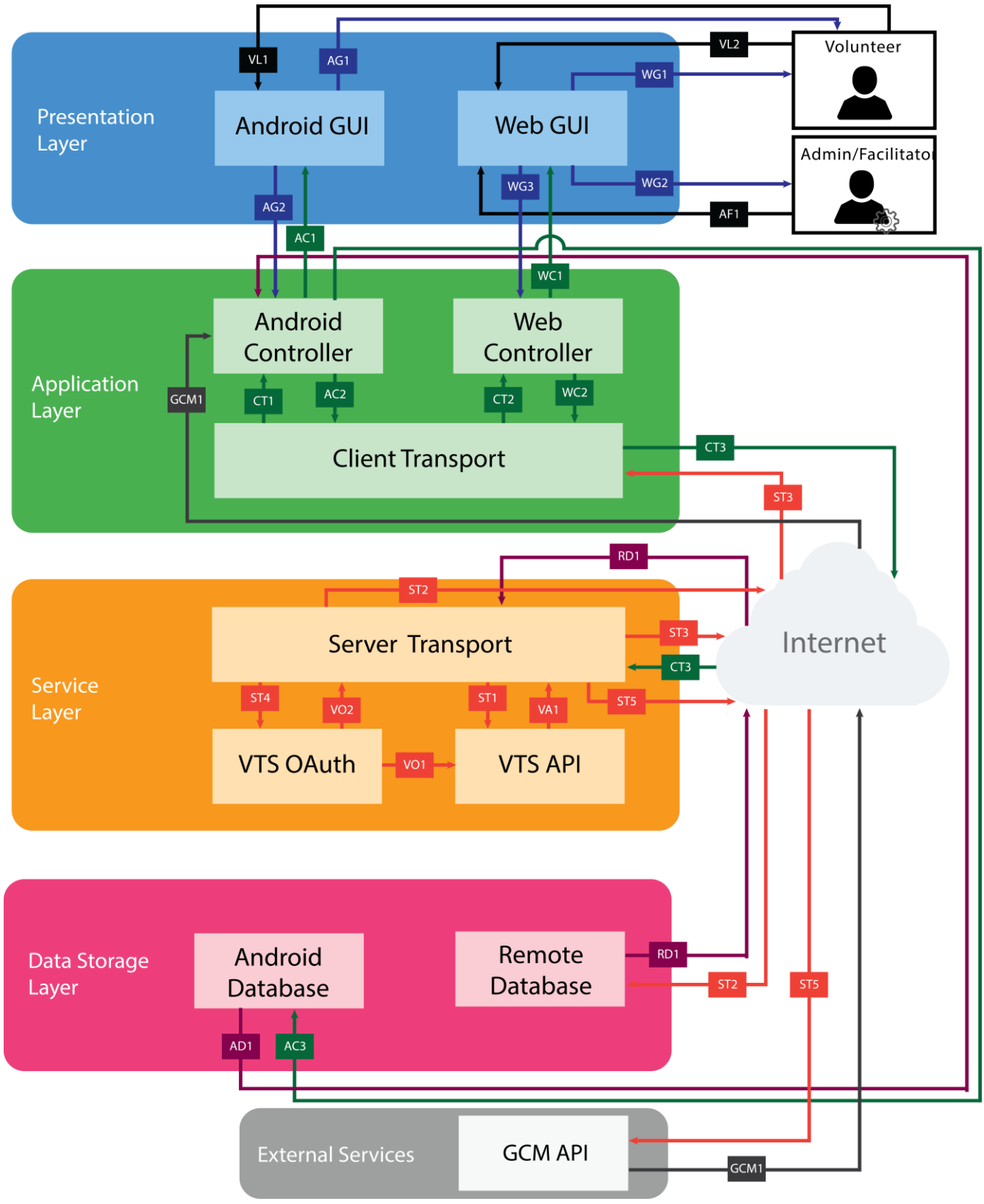


Figure 4.1 Data Flow Diagram

4.2 Data Flow Definitions

The below table illustrates the type or the state of the data flowing throughout different sub systems.

Data Element	Data Type	Description
AC1	Formatted Data	Based on the information passed through the Client Transport Subsystem, Android Controller Subsystem formats the data so that it can be presented to the user via the Android GUI Subsystem.
AC2	JSON Request	The Android Controller Subsystem converts or formats the data so it can be easily transferred to the Service Layer via the Client Transport Subsystem
AC3	Database Query	According to the user interactions and the response from the GCM API, Android Controller Subsystem passes a Database Query to its internal Android Database Subsystem.
AD1	RAW Data	RAW Data coming from the Internal Database in the Android App that are not ready to pass it to the Android GUI Subsystem
AF1	User Input	Admin/Facilitator interacts with the Web App via mouse clicks and selections
AG1	Visual Response	Based on the Volunteer input or the server response, Android App changes its view or display a visual indicator to reflect the respective changes
AG2	RAW Data	Based on the user interaction, Android GUI passes RAW information such as tapped button ids, selection indices etc. to the Android Controller
CT1	RAW Data	Client Transport Subsystem transfers the data coming from the Service Layer to the Android Controller without processing
CT2	RAW Data	Client Transport Subsystem transfers the data coming from the Service Layer to the Web Controller without processing
CT3	HTTP Request	Android Controller and Web Controller access the Remote Database via the VTS API; this request is made via HTTP GET or HTTP POST

GCM1	JSON Response	Google Cloud Messaging servers notifies of the updated content in the Remote Database Subsystem via a JASON response to the registered Android devices. The response includes unique device IDs and information regarding the updated content, date and time.
RD1	RAW Data	Remote Database provides RAW Data to the Server Transport Subsystem based on the received database queries
ST1	RAW Data	Server Transport sends the unformatted/unprocessed data into the VTS API for data processing
ST2	Database Query	As per Android Controller or the Web Controller's requests that are coming from the Application Layer, Server Transport Subsystem makes a Database Query to the Remote Database Subsystem
ST3	HTTP Response	The response coming from the VTS API is passed to the Application Layer via the Server Transport via HTTP
ST4	RAW Data	Server Transport sends the unformatted data such as username and hashed passwords to the VTS OAuth Subsystem
ST5	HTTP Request	Whenever a change has been made in the Remote Database, Server Transport Subsystem notify of that change via the Google Cloud Messaging Service; this request is made via HTTP
VA1	JSON Response	VTS API Subsystem exports the data received from the Remote Database Subsystem allowing services and apps external to the server such as the Android App, Web App or GCM can access the system data
VL1	User Input	Volunteer interacts with the Android App via taps, selections and swipes
VL2	User Input	Volunteer interacts with the Web App via mouse clicks and selections
VO1	JSON Response	VTS OAuth Subsystem checks the user information provided by the Server Transport Subsystem for its validity and passes the response which can be either "valid" or "invalid" to the VTS API
VO2	JSON Response	VTS OAuth sends the status of the user validation to the Server Transport Subsystem in a JSON form

WC1	Formatted Data	Based on the information passes through the client Transport Subsystem, Web Controller outputs the data in a format that can be represented via the Web GUI Subsystem
WC2	AJAX Request	Web Controller requests the data from the Remote Database and this request is made to the Client Transport as an AJAX Request
WG1	Visual Response	Based on the Volunteer input or the server response, Web App changes its view or display a visual indicator to reflect the respective changes
WG2	Visual Response	Based on the Admin/Facilitator input or the server response, Web App changes its view or display a visual indicator to reflect the respective changes
WG3	RAW Data	Based on the user interaction, Web GUI transfers the RAW Data such as on click event information, selection indices etc. to the Web Controller

Table 4.2 Data Flow Definitions

4.3 Producer-Consumer Relationships

The following matrix depicts how data flows between different subsystems. From this matrix we can see that the data flows are balanced as no subsystem is consuming more than four elements and no subsystem is producing more than five elements. The dispersion of data elements can be seen, as they are not heavily clustered in one area of the matrix. Based on this analysis, we can conclude that our architecture is modular where each subsystem plays an integral part of the architecture and is not being burdened with additional responsibilities.

	Consumer	Volunteer	Admin/Facilitator	Android GUI	Web GUI	Android Controller	Web Controller	Client Transport	Server Transport	VTS OAuth	VTS API	Android Database	Remote Database	GCM API
Producer														
Volunteer				VL1	VL2									
Admin/Facilitator					AF1									
Android GUI		AG1				AG2								
Web GUI		WG1	WG2				WG3							
Android Controller				AC1			AC2					AC3		
Web Controller					WC1			WC2						
Client Transport					CT1		CT2		CT3					
Server Transport								ST3		ST4	ST1		ST2	ST5
VTS OAuth											VO1			
VTS API									VA1					
Android Database						AD1								
Remote Database									RD1					
GCM API						GCM1								

Table 4.3 Producer-Consumer Relationships

5. Presentation Layer

5.1 Web GUI Subsystem

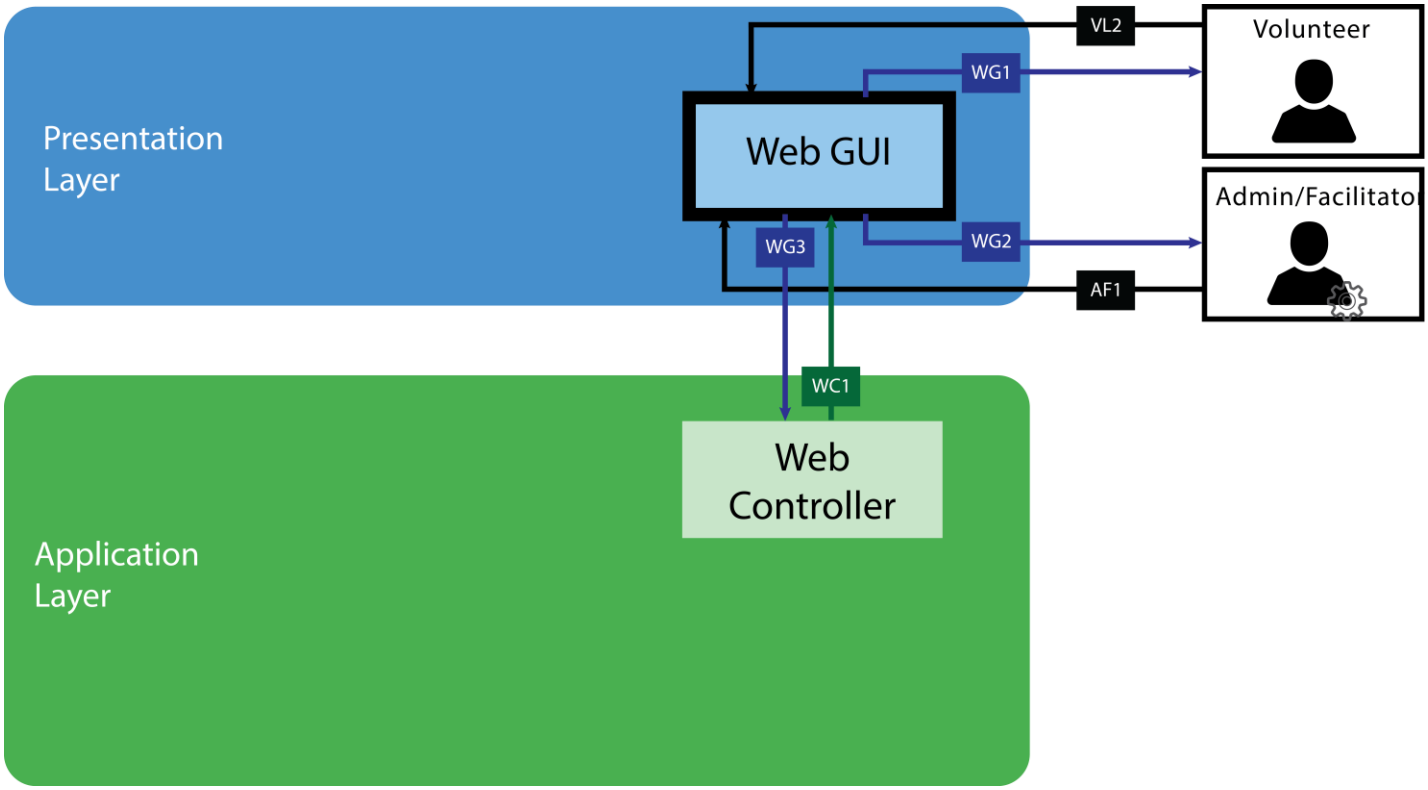


Figure 5.1 Web GUI Subsystem

5.1.1 General Description: The Web GUI Subsystem will receive input from the user (volunteers, facilitators, or administrators) and send that input to the Application Layer. This subsystem will also retrieve data from the Application Layer and present the data to the user.

5.1.2 Assumptions: The Web GUI will be able to handle all necessary data input from the user.

5.1.3. Responsibilities: The Web GUI Subsystem will be responsible for handling input from the user and displaying data from the Remote Database System via the Web Controller Subsystem in the Application Layer. User input may include data to be stored in the Remote Database or a request to view particular data in the Remote Database.

5.1.4. Inter-Layer Interfaces:

Method	Description	Information Required	Information Returned
dataFormattedListener	The Web GUI will receive formatted data in the form of a string or an array from the Web Controller to display to the user	None	Formatted Data
sendInput	Based on the user interaction, Web GUI transfers the RAW Data such as onclick event information, selection indices etc. to the Web Controller	User RAW input data	None

5.1.5 Public Interfaces:

Method	Description	Information Required	Information Returned
userInputListener	Listens for input from the user – the Admin/Facilitator or the Volunteer and in most cases this will be a mouse click	User input either as string or integer and a tokenized string	None
displayData	Displays data received from the Web Controller Subsystem through the website	Formatted data	None

5.2 Android GUI Subsystem

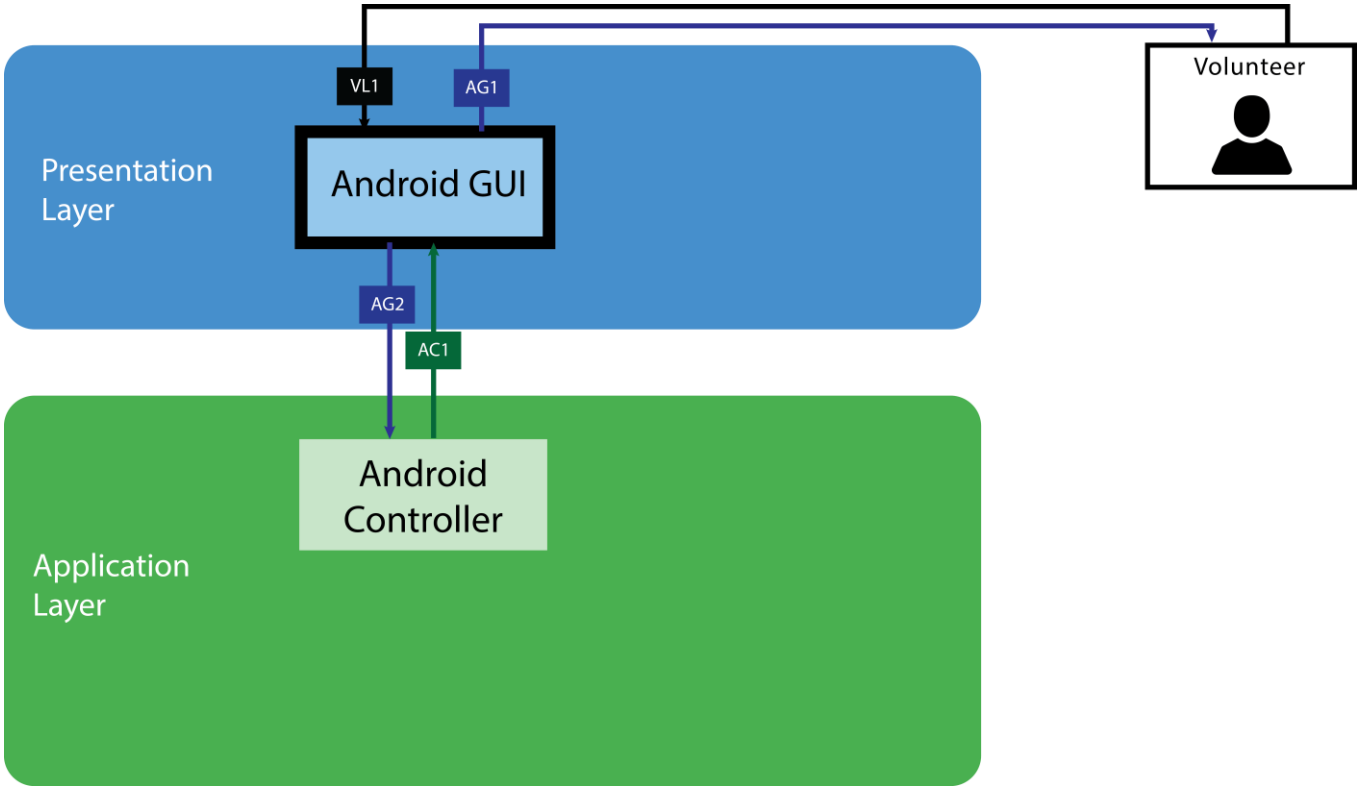


Figure 5.2 Android GUI Subsystem

5.2.1 General Description: The Android GUI Subsystem will receive input from the user (only volunteers) and send that input to the Application Layer. This subsystem will also retrieve data from the Application Layer and present the data to the user.

5.2.2 Assumptions: The Android GUI will be able to handle all necessary data input from the user.

5.2.3. Responsibilities: The Android GUI Subsystem will be responsible for handling input from the user and displaying data from the Android Database Subsystem via the Android Controller Subsystem in the Application Layer. User input may include data to be stored in the Android Database or a request to view particular data in the Android Database.

5.2.4. Inter-Layer Interfaces:

Method	Description	Information Required	Information Returned
dataFormattedListener	The Android GUI will receive formatted data from the Android Controller to display to the user	None	Formatted Data
sendInput	Based on the user interaction, Android GUI passes RAW information such as tapped button ids, selection indices etc.	User RAW input data	None

5.2.5 Public Interfaces:

Method	Description	Information Required	Information Returned
userInputListener	Listens for input from the Volunteer via taps, selections, or swipes	User input	None
displayData	Based on the Volunteer input or the server response, Android App changes its view or display a visual indicator to reflect the respective changes	Visual Response	None

6. Application Layer

6.1 Android Controller Subsystem

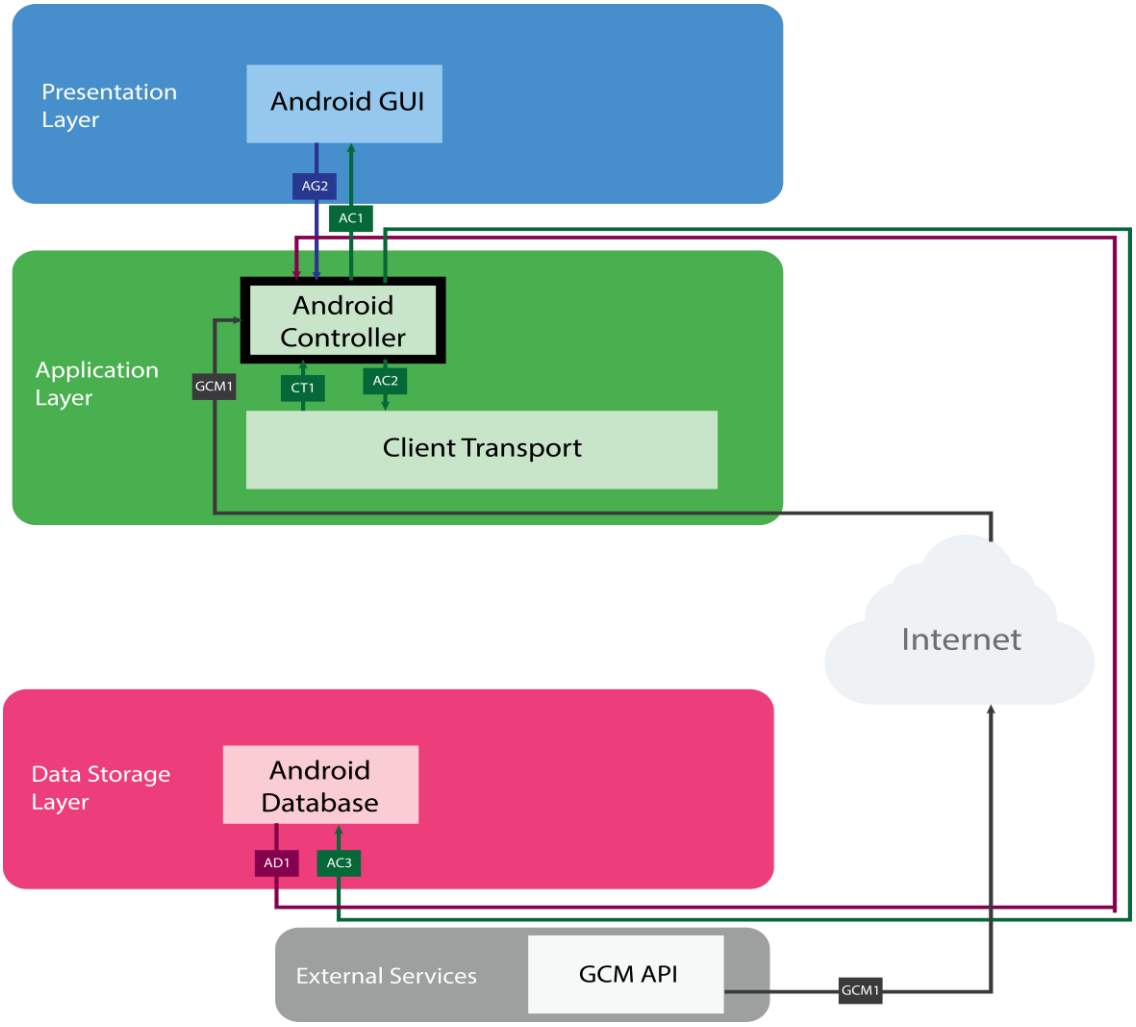


Figure 6.1 Android Controller Subsystem

6.1.1 General Description: The Android Controller Subsystem will send data to the Presentation Layer from the Android Database Subsystem directly and Remote Database indirectly via the Client Transport Subsystem. It will also send user input data to the Client Transport Subsystem.

6.1.2 Assumptions: Internet access is required in order for the Android Controller to process notifications received from the GCM API.

6.1.3. Responsibilities: The Android Controller Subsystem will receive data from the Android Database and send that data to the Android GUI Subsystem. The GCM API will interact with the Android Controller when a notification needs to be sent to the Android GUI. The Client Transport Subsystem will also send data to the Android Controller. This data will ultimately come from the Remote Database System. The data from the Android Database and Client Transport will be sent to the Android GUI. User input data received from the Android GUI will be sent to the Client Transport Subsystem, which will ultimately interact with the Remote Database Subsystem. The Android Controller will also send a token, which is the user's device ID, to the Client Transport that will be used during the authentication process.

6.1.4. Inter-Layer Interfaces:

Method	Description	Information Required	Information Returned
sendFormattedData	Based on the information passed through the Client Transport Subsystem, Android Controller formats the data so that it can be presented to the user via the Android GUI	Formatted Data	None
receiveRAWData	Based on the user interaction, Android Controller receives the RAW information such as tapped button ids, selection indices etc and processes it.	None	User Input Data
receiveDatabaseData	The Android Controller receives RAW Data coming from the Internal Database in the Android App	None	RAW Data

performDatabaseQuery	According to the user interactions and the response from the GCM API, Android Controller Subsystem passes a Database Query to its internal Android Database	Database Query	None

6.1.5 Public Interfaces:

Method	Description	Information Required	Information Returned
receiveGCMData	Google Cloud Messaging servers notifies of the updated content in the Remote Database Subsystem via a JSON response to the registered Android devices. This information is passed to the Android Controller.	None	JSON Object

6.2 Web Controller Subsystem

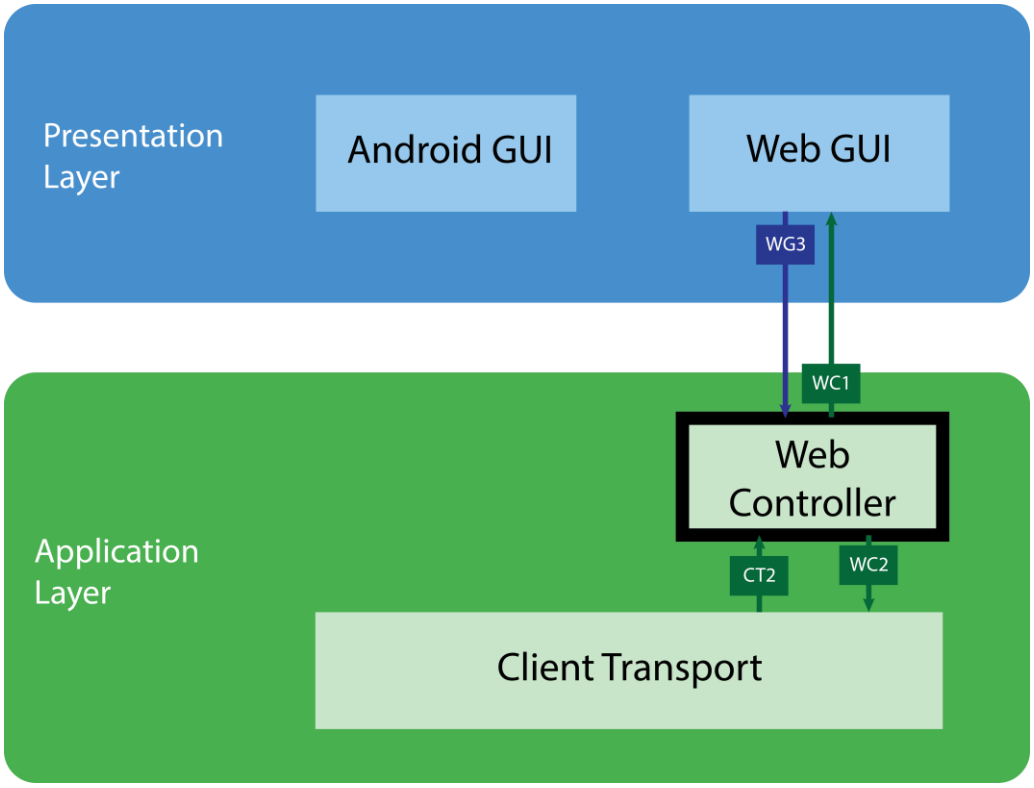


Figure 6.2 Web Controller Subsystem

6.2.1 General Description: The Web Controller will provide the Web GUI with data requested by the user based on user interaction with the Web GUI. The Web Controller will also send user input data to the Client Transport Subsystem.

6.2.2 Assumptions: It is assumed that the Client Transport Subsystem will provide the Web Controller Subsystem the correct data to be sent to the Web GUI.

6.2.3. Responsibilities: The Web Controller Subsystem will receive user input data from the Web GUI Subsystem and send that data plus a token which will be the user's session ID to the Client Transport Subsystem. The Web Controller will also receive data from the Client Transport and send that data to the Web GUI. This data will be formatted data received from the Remote Database.

6.2.4. Inter-Layer Interfaces:

Method	Description	Information Required	Information Returned
receiveRawData	Based on the user interaction, Web GUI transfers the RAW Data such as onclick event information, selection indices etc. to the Web Controller	None	Raw Data
sendFormattedData	Based on the information passed through the Client Transport Subsystem, Web Controller outputs the data in a format that can be represented via the Web GUI Subsystem	Formatted Data	None
sendAJAXRequest	Web Controller requests the data from the Remote Database and this request is made to the Client Transport as an AJAX Request	AJAX Request	None
receiveProcessingData	Client Transport Subsystem transfers data coming from the Service Layer to the Web Controller to format data	None	RAW Data

6.2.5 Public Interfaces: None

6.3 Client Transport Subsystem

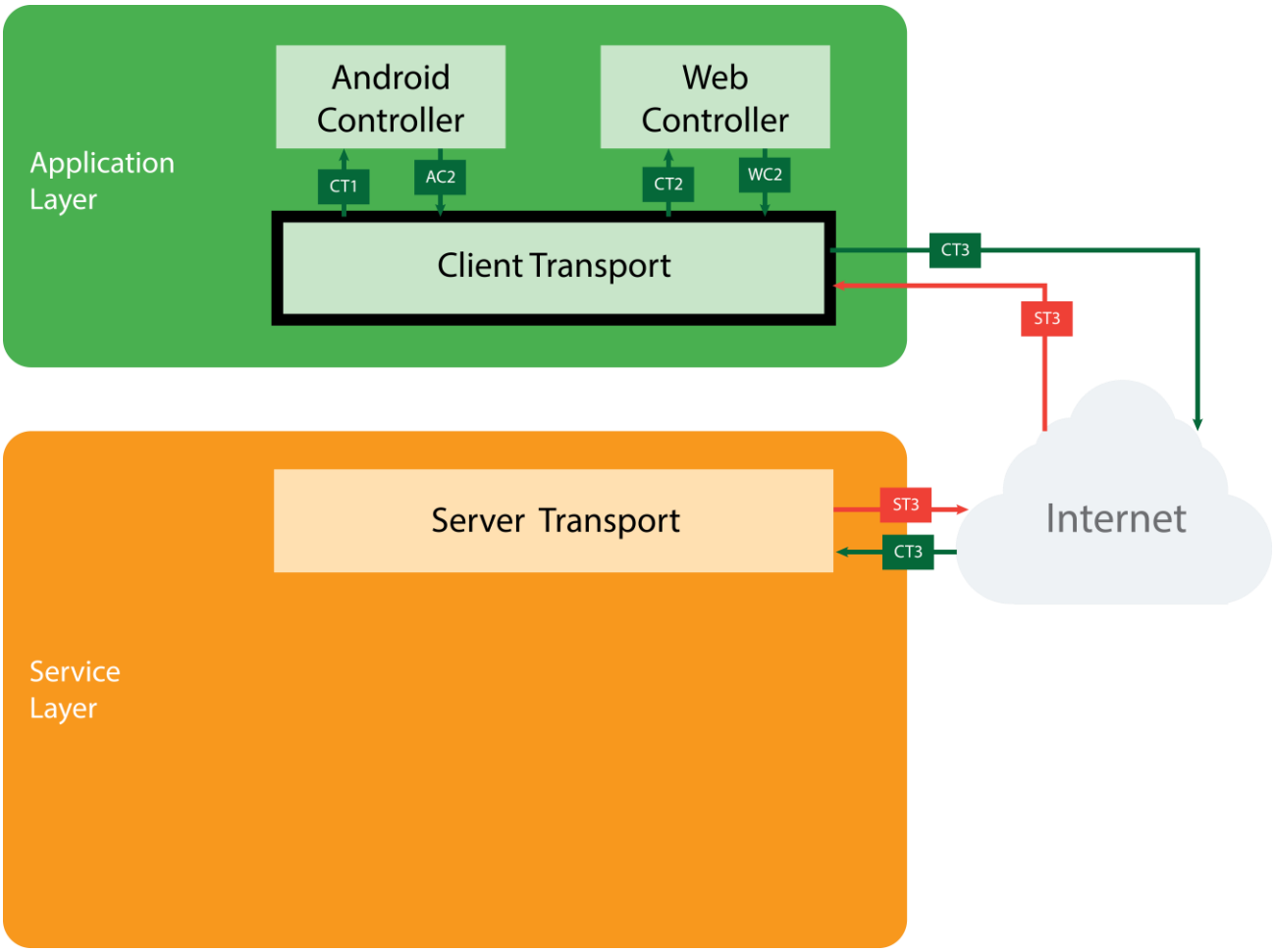


Figure 6.3 Client Transport Subsystem

6.3.1 General Description: The Client Transport Subsystem receives data from the Server Transport Subsystem and sends that data to the Android Controller or Web Controller. The Client Transport also receives data from the Android Controller and Web Controller and sends that data to the Server Transport.

6.3.2 Assumptions: Internet access is required to send data to and from the Client Transport and the Server Transport Subsystems.

6.3.3. Responsibilities: The Client Transport Subsystem will receive data from the Android and Web Controllers. This data will be user input and user interaction data as well as a token for authorization. This data will be sent to the Service Layer. The Server Transport Subsystem

will also send output data (data to be shown to the user) to the Client Transport. This data will come from the Remote Database.

6.3.4. Inter-Layer Interfaces:

Method	Description	Information Required	Information Returned
receiveAPIResponse	The response coming from the VTS API is passed to the Application Layer via the Server Transport via HTTP	None	HTTP Response
sendAccessRequest	Android Controller and Web Controller access the Remote Database via the VTS API; this request is made via HTTP GET or HTTP POST	HTTP Request	None
sendRAWData	Client Transport Subsystem transfers the data coming from the Service Layer to the Android Controller	RAW Data	None
receiveFormattedData	Client Transport Subsystem receives the formatted data so it can easily be transferred to the Service Layer	None	JSON Object
sendProcessingData	Client Transport Subsystem transfers the data coming from the Service Layer to the Web Controller to process it	RAW Data	None

receiveAJAXRequest	Web Controller requests the data from the Remote Database and this request is made to the Client Transport as an AJAX Requests	None	AJAX Request

6.3.5 **Public Interfaces:** None

7. Service Layer

7.1 Server Transport Subsystem

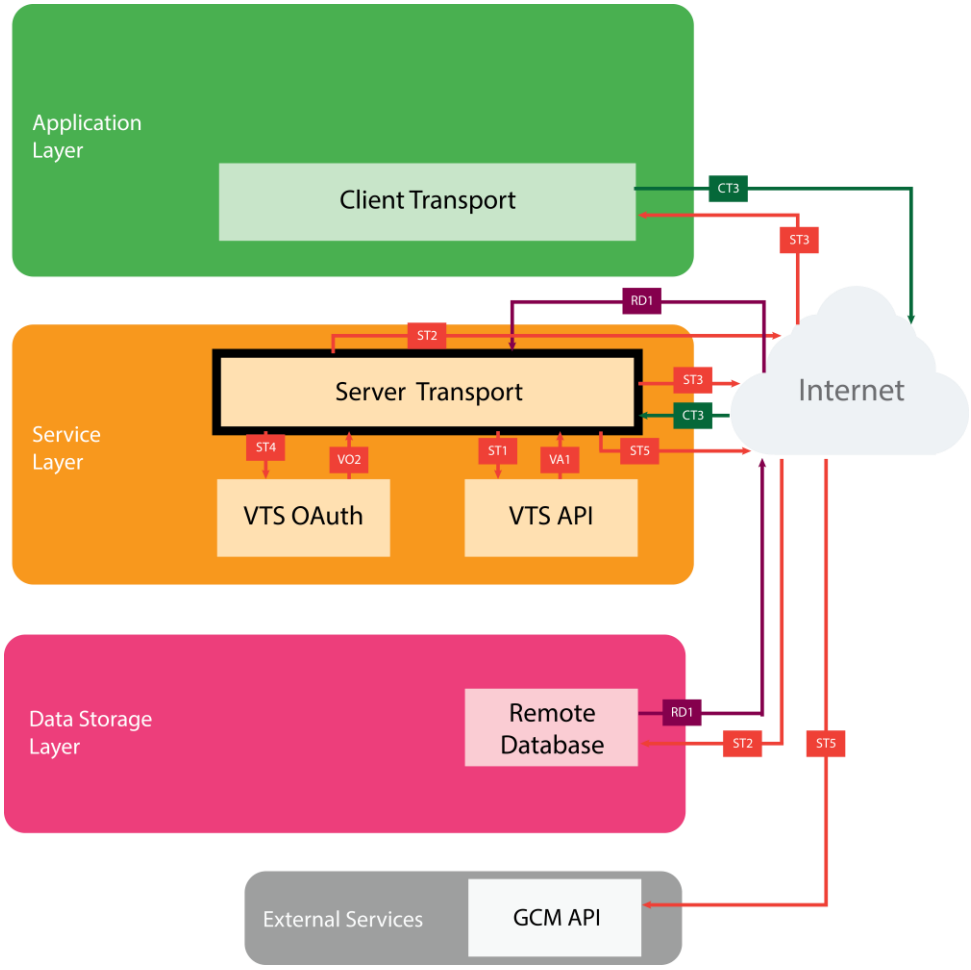


Figure 7.1 Server Transport Subsystem

7.1.1 General Description: The Server Transport Subsystem sends and receives data to and from the Application Layer. The Server Transport also sends and receives data to and from the Remote Database System in the Data Storage Layer. Data is sent to the GCM API from the Server Transport.

7.1.2 Assumptions: Internet access is required for the Application Layer and Server Transport Subsystem to communicate.

7.1.3. Responsibilities: The data received from the Client Transport Subsystem will be sent to the VTS OAuth and VTS API Subsystems which process that data. That data is then sent back to the Server Transport Subsystem which is responsible for sending it to the Remote Database Subsystem. The Remote Database Subsystem sends new data back to the Server Transport which is then responsible for sending that data to the Client Transport via the VTS API. The Server Transport is also responsible for sending data to the GCM API. This data is used to check if a notification must be sent to the user via the Application and Presentation Layers.

7.1.4. Inter-Layer Interfaces:

Method	Description	Information Required	Information Returned
sendDatabaseQuery	As per Android Controller or the Web Controller's requests that are coming from the Application Layer, Server Transport Subsystem makes a Database Query to the Remote Database Subsystem	Database Query	None
sendHTTPReponse	The response coming from the VTS API is passed to the Application Layer via the Server Transport via HTTP	HTTP Response	None
receiveHTTPRequest	Android Controller and Web Controller access the Remote Database via the VTS API; this request is made via HTTP GET or HTTP POST	None	HTTP Request

receiveRawData	Remote Database provides RAW Data to the Server Transport Subsystem based on the received database queries	None	RAW Data
sendRawAuthData	Server Transport sends the unformatted data such as username and hashed passwords to the VTS OAuth Subsystem	Raw Data	None
sendUnformattedData	Server Transport sends the unformatted/unprocessed data into the VTS API for data processing	Unformatted Data	None
receiveValidationStatus	VTS OAuth sends the status of the user validation to the Server Transport Subsystem in a JSON form	None	JSON Object
receiveProcessingData	VTS API Subsystem exports the data received from the Remote Database Subsystem to the Server Transport Layer	None	JSON Object

7.1.5 Public Interfaces:

Method	Description	Information Required	Information Returned
sendChangeRequest	Whenever a change has been made in the Remote Database, Server Transport Subsystem notify of that change via the Google Cloud Messaging Service; this request is made via HTTP	HTTP Request	None

7.2 VTS OAuth Subsystem

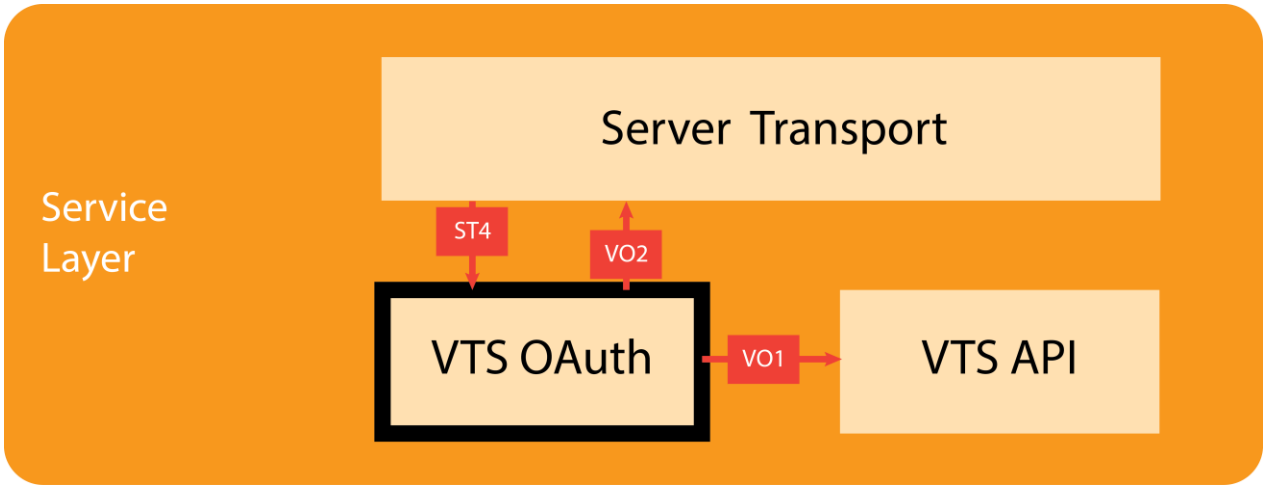


Figure 7.2 VTS OAuth Subsystem

7.2.1 General Description: The VTS OAuth Subsystem will authorize any changes made to the system.

7.2.2 Assumptions: The user has allowed the Android Application to use GCM. Also, the Android Application will be connected to the internet.

7.2.3. Responsibilities: The VTS OAuth is responsible for making sure that when an attempt is being made to change the Remote Database, the user making that change is authorized to do so. It will receive user input data from the Server Transport Subsystem and check whether the user input data contains a valid token. If so, it sends the user input data to the VTS API Subsystem to further process the data. If it is unsuccessful in authorizing the user, then it will send a JSON message to the Server Transport that will end up being displayed to the user.

7.2.4. Inter-Layer Interfaces:

Method	Description	Information Required	Information Returned
receiveRawAuthData	Server Transport sends the unformatted data such as username and hashed passwords to the VTS OAuth Subsystem	None	Raw Data
sendValidationStatus	VTS OAuth sends the status of the user validation to the Server Transport Subsystem in a JSON form	JSON Object	None
sendAuthResponse	VTS OAuth Subsystem checks the user information provided by the Server Transport Subsystem for its validity and passes the response which can be either "valid" or "invalid" to the VTS API	JSON Response	None

7.2.5 Public Interfaces: None

7.3 VTS API Subsystem

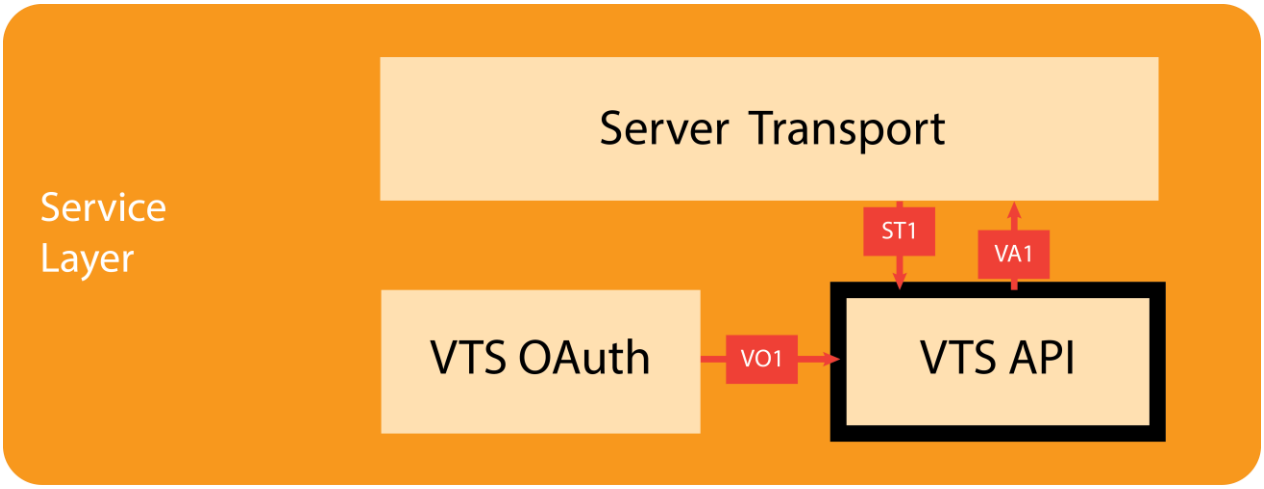


Figure 7.3 VTS API Subsystem

7.3.1 General Description: The VTS API Subsystem will send and receive data to and from the Server Transport Subsystem. The data sent to the Server Transport will either get sent to the Application Layer (output) or the Data Storage Layer (input).

7.3.2 Assumptions: There is enough space on the user's Android device to store an Internal Database.

7.3.3. Responsibilities: The three main responsibilities of the VTS API are to format the data received from the Server Transport Subsystem (user input) into JSON objects, input data into the Remote Database via the Server Transport, and retrieve data from the Remote Database, again via the Server Transport.

7.3.4. Inter-Layer Interfaces:

Method	Description	Information Required	Information Returned
receiveJSONResponse	VTS OAuth Subsystem checks the user information provided by the Server Transport Subsystem for its validity and passes the response to the VTS API	None	JSON Response
receiveProcessingData	Server Transport sends the unformatted/unprocessed data into the VTS API for data processing	None	Raw Data
sendProcessingData	VTS API Subsystem sends the data received from the Remote Database Subsystem to the Server Transport Subsystem	JSON Object	None

7.3.5 Public Interfaces: None

8. Data Storage Layer

8.1 Android Database Subsystem

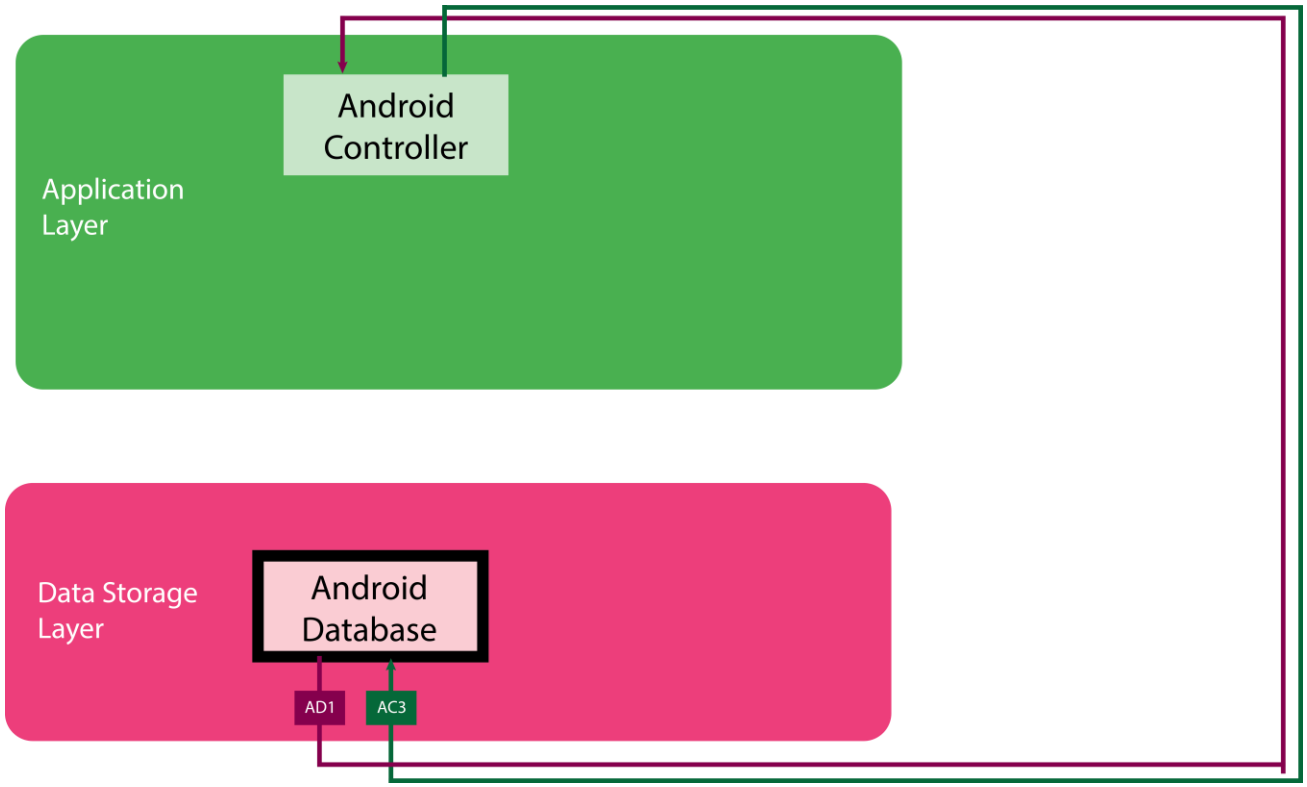


Figure 8.1 Android Database Subsystem

8.1.1 General Description: The Android Database Subsystem will store all of the data required for the Android Application Subsystem. The Android Controller Subsystem will be able to access data from the Android Database Subsystem without needing to be connected to the internet.

8.1.2 Assumptions: There is enough space on the user's Android device to store an internal database.

8.1.3. Responsibilities: The sole responsibility of the Android Database Subsystem is to store user (volunteer) data needed for the Application Layer. The Android Database Subsystem allows users to use the Android application without connecting to the internet.

8.1.4. Inter-Layer Interfaces:

Method	Description	Information Required	Information Returned
sendDatabaseQuery	According to the user interactions and the response from the GCM API, Android Controller Subsystem passes a Database Query to its internal Android Database	Database Query	None
receiveRawData	RAW Data coming from the Internal Database in the Android App that are not ready to pass it to the Android GUI Subsystem	None	Raw Data

8.1.5 Public Interfaces: None

8.2 Remote Database Subsystem

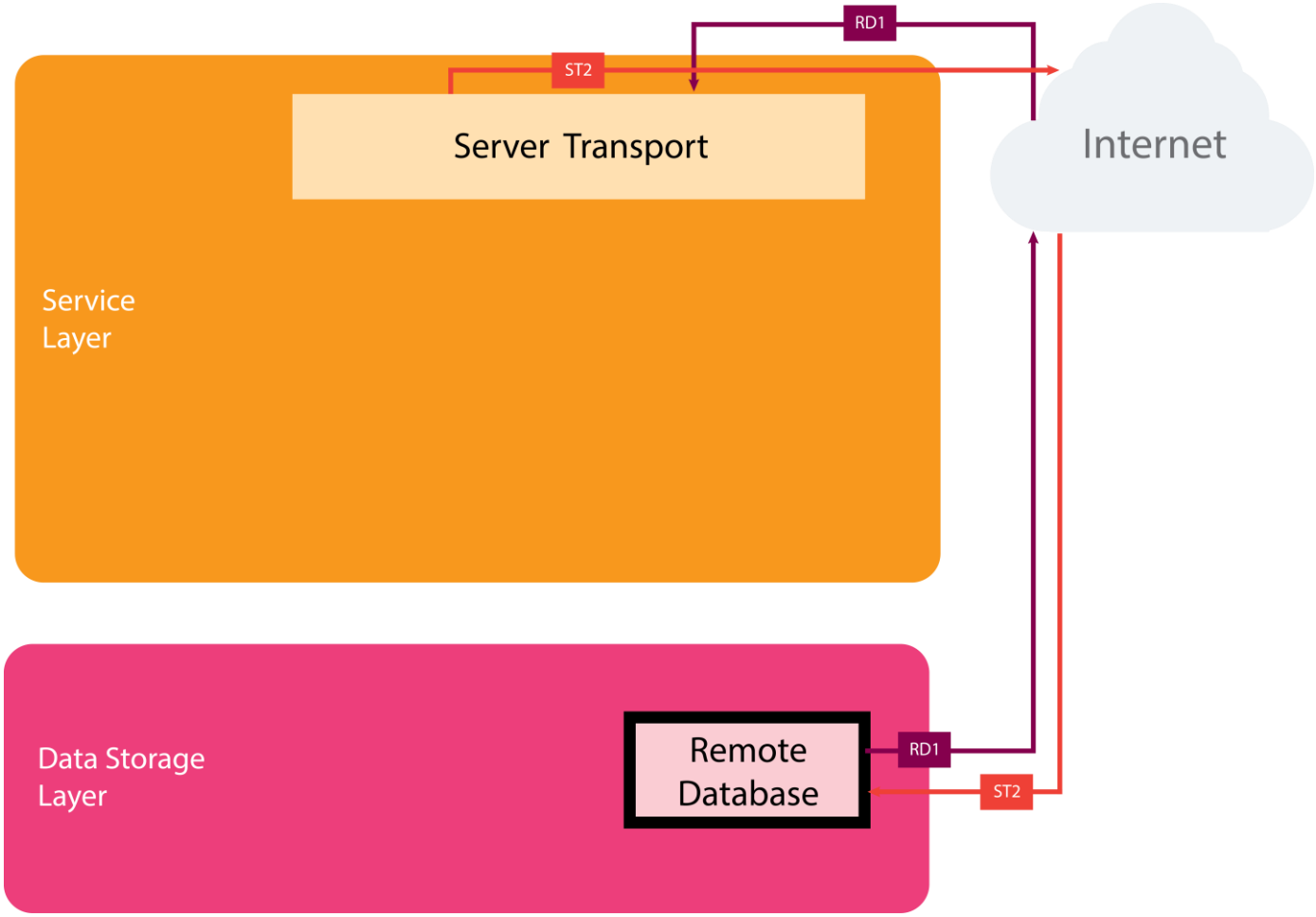


Figure 8.2 Remote Database Subsystem

8.2.1 General Description: The Remote Database Subsystem will store all of the system's data. The Server Transport Subsystem will send data to and from the Remote Database.

8.2.2 Assumptions: The user has allowed the Android Application to use GCM. Also, the Android Application will be connected to the internet.

8.2.3. Responsibilities: The Remote Database Subsystem is responsible for storing all of the information needed for the system. Most of this information will be related to volunteer tracking and events management. Other information will be related to security, such as login credentials, device IDs, and session IDs. The Remote Database is used indirectly by the Web Controller Subsystem and Android Controller Subsystem via the Service Layer.

8.2.4. Inter-Layer Interfaces:

Method	Description	Information Required	Information Returned
receiveDatabaseQuery	As per Android Controller or the Web Controller's requests that are coming from the Application Layer, Server Transport Subsystem makes a Database Query to the Remote Database Subsystem	None	Database Query
sendRawData	Remote Database provides RAW Data to the Server Transport Subsystem based on the received database queries	Raw Data	None

8.2.5 Public Interfaces: None

9 Requirements Traceability

This section illustrates which layer fulfills each requirement. By having a clear vision of how each layer contributes to fulfill the overall system requirement, we will be able to design our system in a better way.

9.1 Requirements Traceability Matrix

Requirement Number	Requirement Name	Layer			
		Presentation	Application	Service	Data Storage
3.1	Input Volunteer Hours	✓			✓
3.2	Notify Admin	✓	✓	✓	
3.3	Input Volunteer Hours on behalf of user	✓			✓
3.4	Add Volunteer Opportunities	✓			✓
3.5	Delete Volunteer Opportunities	✓			✓
3.6	Sign-Up for Volunteer Opportunities	✓	✓	✓	
3.7	Cancel Commitment	✓	✓	✓	
3.8	Notify Volunteer	✓	✓	✓	
3.9	Track Progress	✓	✓	✓	
3.10	Generate Reports	✓	✓	✓	
3.14	Customize Preferences	✓			✓
3.15	Login	✓	✓	✓	
3.16	Logout	✓	✓	✓	

3.18	Register Volunteers	✓			✓
3.22	Ease of use	✓			
3.23	Android Application	✓	✓	✓	✓
6.2	Password Encryption	✓	✓		✓

Table 8.1 Requirements Traceability Matrix

9.2 Requirements Traceability Analysis

As we can see from the above table, most of the responsibility is handled by the Presentation and Application layers. This is an indication that our layer design is appropriate since our system is extremely interactive. The user will only interact with the system through the GUI interfaces so the Presentation layer is required for most of the requirements. Some of the requirements such as generating reports and tracking progress require extensive data processing and formatting so these requirements are fulfilled by the Application layer in conjunction with the Service layer. The Data Storage layer does not have excessive responsibilities since it is only responsible for storing and allowing easy access to the data via the Service layer. Hence, it is not extensively involved in the fulfillment of most of the key requirements.

We can confidently say that our layers are independent to each other and fulfill a specific purpose. The independent layers allow for a highly modular system that can be easily tested and extended as required in the future.

10. Operating System Dependencies

This section describes uses of any libraries, operating systems, and APIs in each layer. The system is designed in such a way that each layer performs a well-defined function.

10.1 Presentation Layer

The Android GUI of Presentation Layer will be dependent on the Android Operating System, minimum API level of 16, which corresponds to version 4.1.2 (Jelly Bean) for mobile application. The layer will have additional Android specific libraries that help with the transfer of data between components in the system.

The Web GUI of presentation layer will use the Web Components as the building blocks of web app. The web page creation will dependent on various web technologies such as HTML5, CSS3, JavaScript and Polymer. The Web GUI will be compatible with all major versions of web browsers such as Mozilla Firefox, Google Chrome, Internet Explorer, and Safari released within the last two years.

10.2 Application Layer

The Application Layer consists of Android Controller, Web Controller, and Client Transport. Android API will be used to sync Remote Database and Android database in Android Controller. Android Controller also use relation SQLite database and other APIs to make a direct connection with Android Database and process database query when the system is not connected to internet. Web Controller will make AJAX requests by HTTP GET and HTTP POST methods to transfer data to Client Transport. Both Android and Web apps will use JSON parsing libraries to parse the data before passing them to Client Transport. Client Transport will make HTTP request to transfer data including attachments to service layer.

10.3 Service Layer

The Service Layer will be the server side of the Volunteer Tracking System. The layer consist of Server Transport, VTS OAuth, VTS API, and Web Service. The layer will be responsible to make direct connection with Data Storage Layer. The layer will use PHP PDO library to make a secure MySQL connection. The VTS API will use API's making database query to add, update and retrieve Remote Database. The retrieved data will be formatted and passed to Application Layer via Server Transport replying to HTTP request from Client Transport.

10.4 Data Storage Layer

The Data Storage Layer is mainly responsible for storing data. The Layer has Remote Database and Android Database. The Server Transport subsystem will update and retrieve data only from Remote Database whereas Android Database is used by Android Database when the system is offline. Android Database will be synced with Remote Database via Google Cloud Messaging (GCM) when there is internet connection. Any update on Remote Database will be notified to Android Database for syncing process. The Android Database will be accessed directly from Android Controller in Application Layer by relational SQLite queries.

11. Testing Considerations

This section describes the testing considerations that will be considered in each layer of the system to effectively test the functionality of each layer. The testing considerations will be derived from the guiding principles discussed earlier. Testing is conducted to verify and validate each layer for reliability, intuitiveness, and responsiveness.

11.1 Overall Considerations

11.1.1 Code Review: Code reviews will be held frequently to verify that functions being implemented are corresponding to one or more requirements. Code reviews are useful tools to improve overall performance and to find potential problems or bugs early in the project to avoid bigger problems in a later stage of development.

11.1.2 Ease of Use: The system should be intuitive and easy to use for the user. The user should not be required to know the implementation of a function or have a great deal of knowledge or training to use the system.

11.1.3. Beginning to End: The system will be tested from beginning to end upon completion of use cases. The result will be compared to the expected result to verify the overall system with respect to the requirements.

11.2 Presentation Layer

11.2.1 Modularity: The Presentation Layer must be a stand-alone layer and must not be dependent on the internal subsystems of other layers.

11.2.2 Internal System Interaction: The Presentation Layer has Android GUI and Web GUI as its subsystems. Both Subsystems are independent to each other and they do not have any kind of communication between them.

11.2.3 External System Interaction: The Presentation Layer will be directly interacting with users. The Layer also interacts with Application Layer. It will also receive data from the user and pass it to Application Layer for further processing.

11.3 Application Layer

11.3.1 Modularity: The Application Layer must be a stand-alone layer and must not be dependent on the internal subsystems of other layers.

11.3.2 Internal System Interaction: The Application Layer has Android Controller, Web Controller, and Client Transport. The Android Controller and Web Controller do not communicate with each other but both subsystems will be communicate with the Client Transport Subsystems to pass formatted data to the Service Layer for further processing.

11.3.3. External System Interaction: The Application Layer will interact with all other layers of the system. The Layer will be getting raw input from Presentation Layer and will also transfer results to the Presentation Layer for display. The Application Layer will interact with the Database Storage Layer through the Android Controller Subsystem where it will have direct connection with the Android Database Subsystem when the system is offline. The Client Transport will make the HTTP request to pass data to Service Layer and will also receive data from Service Layer.

11.4 Service Layer

11.4.1 Modularity: The Service Layer must be a stand-alone layer and must not be dependent on the internal subsystems of other layers.

11.4.2 Internal System Interaction: The Service Layer has Server Transport, VTS OAuth, and VTS API. The Server Transport is very active subsystem in this layer and it communicates with every other Subsystem. The formatted data and request from Client Transport will be passed to User OAuth and VTS API through the Server Transport Subsystem.

11.4.3. External System Interaction: The Service Layer interacts with Application Layer and Data Storage Layer. Service Layer receives a HTTP request from the Application Layer and processes the request It then sends it back to the Application Layer. It will also interact with Database Storage Layer through various database queries for updating and retrieving data.

11.5 Data Storage Layer

11.5.1 Modularity: The Data Storage Layer must be a stand-alone layer and must not be dependent on the internal subsystems of other layers.

11.5.2 Internal System Interaction: The primary updates and modification of data will be done in Remote Database Subsystem. The Android database will be used only for an Android device associated with Android app. Remote Database and Android Database do not communicate with each other in Data Storage Layer. The syncing process will be done in Application Layer instead of the Data Storage Layer.

11.5.3. External System Interaction: The Data Management Layer interacts with the Service Layer and the Application Layer. The data in Remote Database Subsystem will be updated and retrieved by various database queries coming from the Service Layer. The Android Database can be accessed via the Android Controller in Application Layer. The syncing between the Remote Database and Android Database will be performed in Android

Controller when GCM notifies the Android devices about any changes in Remote Database in presence of Internet.