GBACARD

System Architecture Design Document (SADD)

Version 1.0

4th of November, 2015

Prepared by: Tobiloba Williams, Theophilus Oghenewogaga

TABLE OF CONTENTS

1) INTRODUCTION

This document describes the architecture of the Gbacard system. It describes:

- A general description of the system
- The logical architecture of software, the layers and top-level components
- The physical architecture of the hardware on which runs the software
- The justification of technical choices made
- The traceability between the architecture and the system requirements

2) SOFTWARE ARCHITECTURE

2.1 Architecture Overview

The system works in a mobile environment. The user may carry his mobile device along with him wherever he may wish to go. Hence, the system was developed taking this into consideration. The user may be anyone who wants to share his or her business cards.

The architecture takes these functionalities into consideration.

1. User authentication
2. Signup
3. Password recovery
4. Generating digital business cards
5. User account management
6. VCF creation
7. Sharing of digital business cards
8. Sharing of contacts
9. User account management

The application will use MySQL and SQLite databases internally and will require continued internet access and GPS connectivity.
The system makes use of the external interface, which is the AWS services such as CloudFront and S3.

2.2 Architecture Design Pattern

The application will be implemented using the *Model-View-Presenter* (MVP) architectural pattern.

2.2.1 Why the MVP pattern?

The MVP pattern developed from the MVC (Model View Controller) architectural pattern, which has been gaining importance over time.
In MVC, we have a problem arising from the fact that Android activities are closely coupled to both interface and data access mechanisms. This leads to the following issues:

1. **Testability** - The controller is tied so tightly to the Android APIs that it is difficult to unit test.
2. **Modularity and Flexibility** - The controllers are tightly coupled to the views. It might as well be an extension of the view. If we change the view, we have to go back and change the controller.
3. **Maintenance** - Over time, particularly in applications with anemic models, more and more code starts getting transferred into the controllers, making them bloated and brittle.

The MVP pattern solves this by combining the activity with the view in MVC (XML UI definition) and creating a new layer called the precentor to update the view.

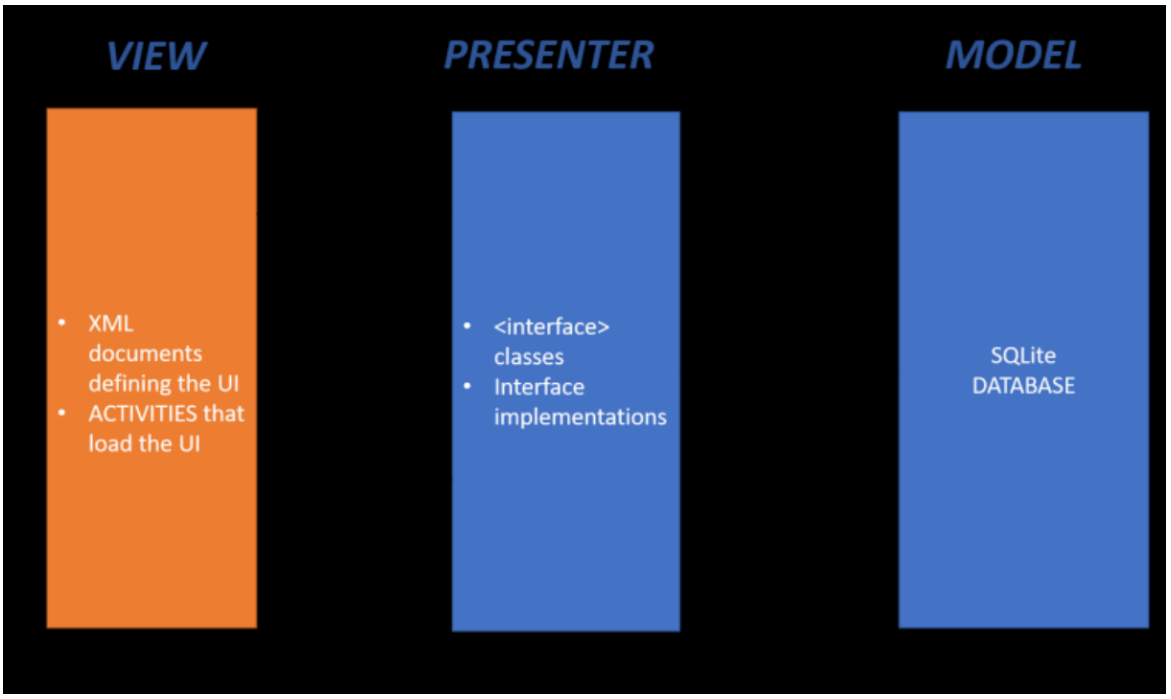

Figure 1 : MVP design

2.3 Logical Architecture Overview

The application logical architecture is developed based on the MVP pattern described above.

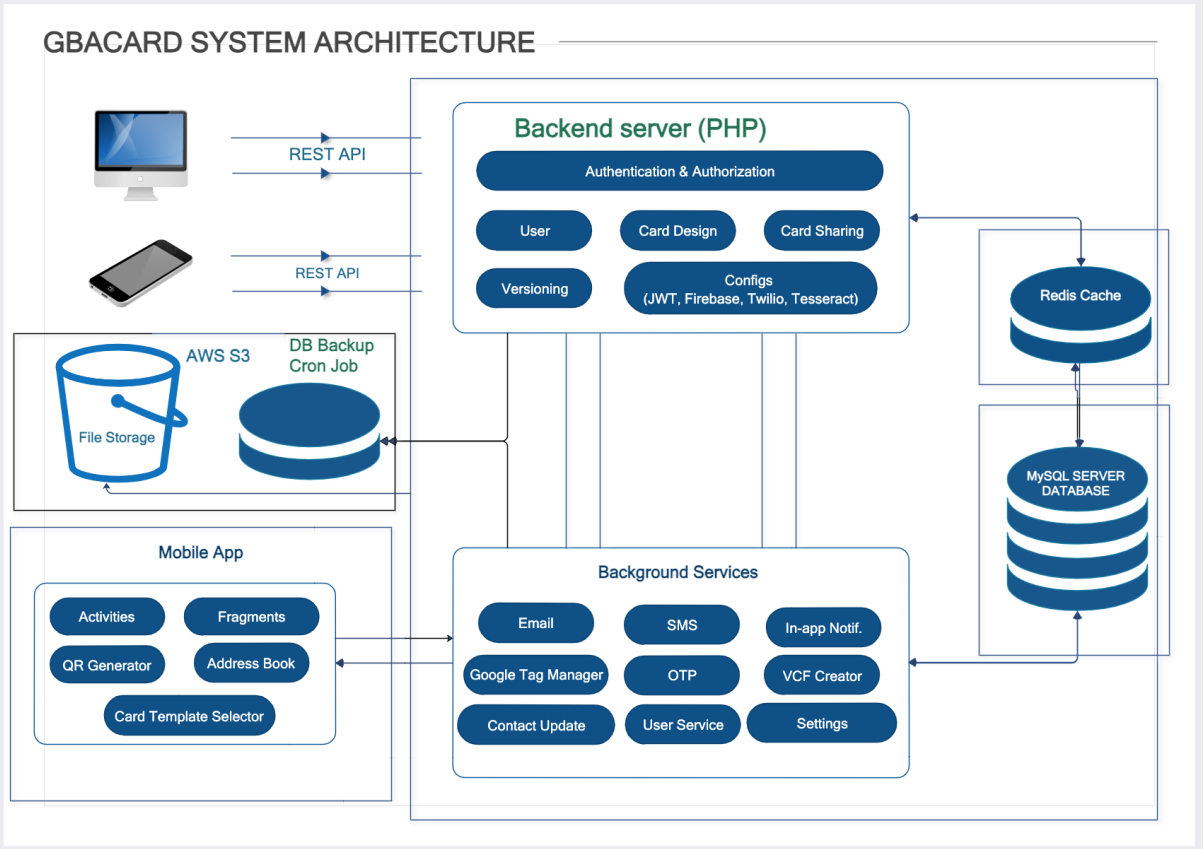

Figure 2: System Architecture

2.3.1.1 View

This layer contains activity classes and XML files that define the structure of the Android application user interface. Each view will also have an interface to link with the presenter. The interface will define the different sections that can be modified in the UI. There are 11 separate activities; the layout of these documents will be defined in the user interface design document.

2.3.1.2 Presenter

This layer defines the classes that hold the core business logic. The presenters form a link between the UI and the model. The interface notifies the presenter based on the user activity, and the presenter will do the necessary logical computation and instruct the user interface to update itself. Each activity will have a corresponding presenter, but the presenter may resort to other helper classes to do some additional computations.

When there is a need to retrieve data from the database, the presenter will make a connection with the SQLite database and view it. The presenter also reads and writes to files in the Android file system. The presenter will also make use of the AWS Mobile SDK to access AWS services.

2.3.1.3 Model

This layer forms a data structure layer, which does all the necessary backend operations related to the business logic.

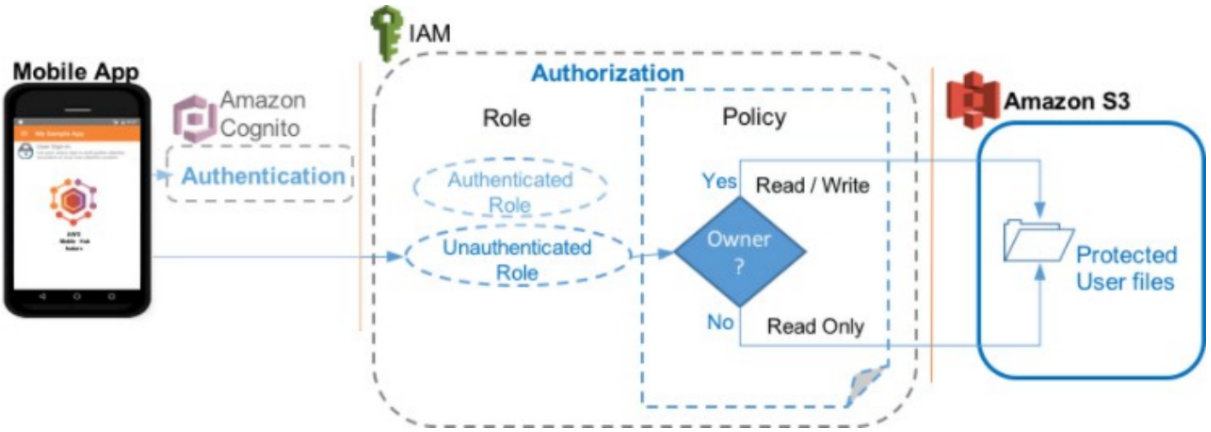

Figure 3: AWS S3 Access

2.3.2 Logical design qualities

The system intends to achieve the following qualities through the above implementation design principles of the application.

2.3.2.1 Modularity

Separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality,

2.3.2.2 High Cohesion

Each module has functions and elements that are strongly related, only to fulfill one purpose or task.

2.3.2.3 Low Coupling

Modules are loosely coupled and independent, so a change in one module does not affect the other modules.

2.3.2.4 Standardization

Implementation will conform to standards that have been established and agreed upon by different parties; this is crucial for things like security.

2.4 Physical Architecture

2.4.1 Physical Architecture Overview

The architecture adopted by the application will be a new and sophisticated *cloud-client architecture*. This architecture works by making use of cloud computing resources to manage user's data and credentials as well as authenticate them. The proposed application will use AWS (Amazon Web Services) as its cloud service provider and the AWS Mobile SDK for the development of the app. The main advantage this architecture has is that we don't need to manage the resources in the cloud, including their security.

2.4.2 Deployment Diagram

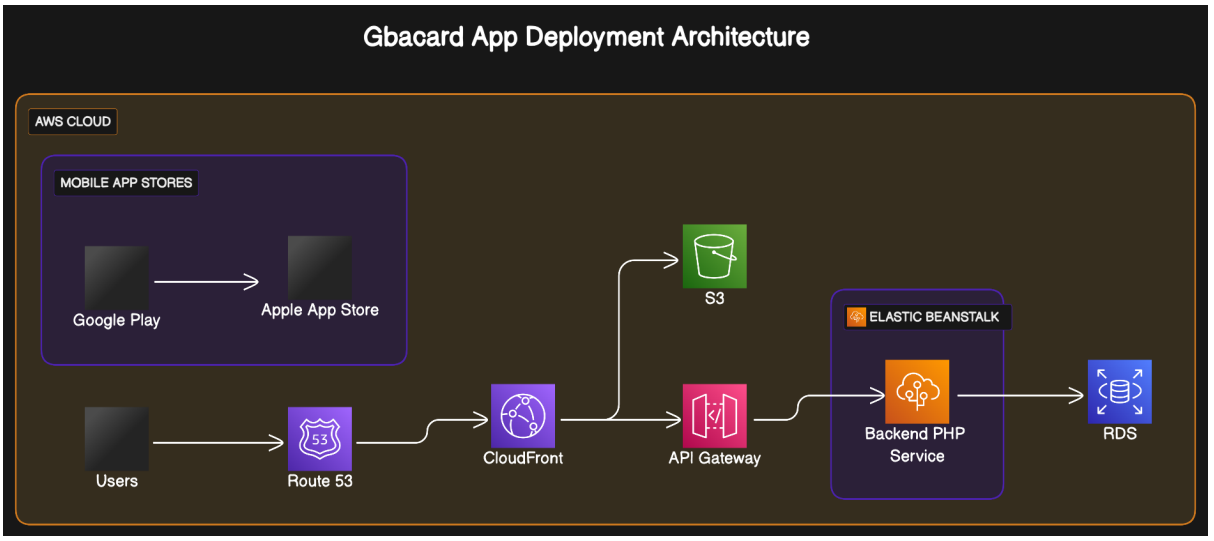

Figure 4: System deployment diagram

The above diagram shows the physical architecture of the system. The execution environment of the system is a mobile device running the Android operating system. The application requires access to the phone's GPS device to get locational details. The application will also have a local SQLite database. Amazon Cloud Service will be the external interface providing centralized user authentication, database, and cloud storage services.

3) SYSTEM ARCHITECTURE QUALITIES

The proposed application will have the following qualities:

3.1.1 Extensibility

The proposed application can add additional functionality without changing or damaging much of the current system. New data types can be added if they are supported by Android.

3.1.2 Maintainability

Following the design principles of high cohesion and low coupling, small modifications will not be a problem. Changing one module will not affect other modules significantly.

3.1.3 Performance

The response time will be acceptable, even with the huge amount of data that is processed. Efficient encryption algorithms are used as well as other processing algorithms.

3.1.4 Usability

Adapting the KISS (Keep It Simple Stupid) principle in designing interfaces will give users easier times in learning and figuring out the proposed application. It lets the user take less time to perform a certain task.

3.1.5 Compatibility

The proposed application will be able to run on various types of Android devices as well as different versions of Android.

3.1.6 Security

Data is kept safe by encryption, and a login is required to have access. Security measures like protection against SQL injection or encryption algorithms will be standard.

4) APPENDIX A: GLOSSARY

- **User Interface (UI):** The front end of the system through which the user interacts with the system functionalities.
- **AWS (Amazon Web Services):** This is a subsidiary of Amazon.com that provides on-demand cloud computing platforms to individuals, companies, and governments on a paid subscription basis.
- **Cognito:** Amazon Cognito is an Amazon Web Services (AWS) product that controls user authentication and access for mobile applications on internet-connected devices.
- **IAM (Identity and Access Management):** AWS Identity and Access Management (IAM) enables you to securely control access to AWS services and resources for your users. Using IAM, you can create and manage AWS users and groups and use permissions to allow and deny their access to AWS resources.
- **MySQL:** MySQL is an open-source relational database management system (RDBMS) that uses structured query language (SQL) to manage and organize data. It is widely used for handling structured data and is known for its reliability, scalability, and performance. MySQL is commonly employed in web applications to store and retrieve data efficiently.
- **Redis:** This is an in-memory data structure store that serves as a key-value database, cache, and message broker. It is highly versatile and performs operations on data structures like strings, hashes, lists, sets, and more. Redis is known for its speed and is often used to improve the performance of applications by caching frequently accessed data.
- **Firebase:** Firebase is a comprehensive mobile and web application development platform provided by Google. It offers a variety of services, including real-time database, authentication, hosting, and cloud functions. Firebase simplifies the development process by providing a set of tools and infrastructure that allows developers to build, test, and deploy applications more efficiently.
- **QR Code Generation:** This refers to the process of creating Quick Response (QR) codes, which are two-dimensional barcodes that store information. These codes can represent various data types, such as text, URLs, or contact information. QR codes are commonly used for quick and efficient data sharing, marketing, and authentication.
- **VCF (vCard File):** VCF, or vCard File, is a standard file format used to store and exchange contact information. It typically contains details such as names, addresses, phone numbers, and email addresses. VCF files are commonly utilized for sharing contact information between applications, devices, and platforms, providing a standardized way to exchange electronic business cards.
- **S3 (Amazon Simple Storage Service):** S3 is a scalable cloud storage service provided by Amazon Web Services (AWS). It allows users to store and retrieve any amount of data at any time through web interfaces or APIs.

S3 is widely used for hosting static websites, storing and retrieving large datasets, and as a content delivery network (CDN) for distributing content globally.

- **Cron Job**: A CRON job is a scheduled task or automated script that runs at specified intervals on Unix-like operating systems. The term "CRON" refers to the time-based job scheduler in Unix-like operating systems. CRON jobs are used to automate repetitive tasks, such as system maintenance, file cleanup, or data backups. Users can set specific time intervals (daily,weekly,or, monthly) for CRON jobs to execute, providing a convenient way to automate routine processes.