

# Developer Design Document

Our project consists of two different applications. One is a front-end application that is intended to be used by medical personnel and their managers. The other is an Android- based smart-phone application that will be primarily used by the workers in the field. Since Android applications are written in Java, we had no choice but to use that for that side of the application. We then decided to implement the front-end in Java as well. Doing so allowed us to take full advantage of the wealth of information of the team, the abundant number of libraries available, the fantastic GUI builders, and most importantly, it allowed us to share common code between the front-end application, and the Android application.

Both applications talk through a centralized platform Windows Azure. We chose to use Windows Azure, due to its large amount of interoperability that was pre-implemented, that we would have otherwise have to implement ourselves. Another nice feature of Windows Azure is that we can have multiple users administrate the services at the same time; something that is necessary when dealing with multiple developers. It also allowed us to easily implement data encryption, as all data transmitted to and from Windows Azure is encrypted with SSL. By doing so, we could skip potentially risky step of encrypting all of our data using a shared public key.

We currently use four services from Windows Azure:

- SQL Database
- Cloud Storage
- Mobile Services
- Website

The SQL Database is our primary database where all records are stored and transactions take place. It utilizes SQL Server in the back-end. We use the Cloud Storage to regularly back up our database in case of accidents, either from a hardware failure, or a temporary lapse in human judgment. The Mobile Service is what is used in order for the Android application to be able to communicate with the SQL Database. Without it, Android lacks the abilities to communicate with an SQL Server-based database. We utilize the website to host a user-registration page running on CodeIgniter, as well as endpoints to the database for the Android application to connect to, which was necessary as the Windows Azure Mobile Services library has issues with tables being touched by anything other than itself.

We use a third-party authentication provider, Auth0, which allows us to easily create and manage multiple users. Using this third party, we had a full API exposed to us that took care of user creation, deletion, key management, and password hashing. We chose Auth0, as it had a web-based API that we could connect with our own scripts and the front-end through traditional HTTP request, as well as an out-of-the-box solution for authenticating with Android devices, linked with Windows Azure Mobile Services.

When authenticating, the user's email address and password is sent to Auth0 for verification. If the username and password is accepted, an ID token is passed back. A typical ID token looks like the following:

```

{
  "iss": "https://login.auth0.com/",
  "sub": "auth0|int",
  "aud": "--",
  "exp": int,
  "iat": int,
  "activated": 1,
  "authlevel": 4,
  "banned": 0,
  "clientID": "--",
  "created_at": "--",
  "email": "--",
  "identities": [ {
    "user_id": int,
    "provider": "auth0",
    "connection": "Username-Password-Authentication",
    "isSocial": false
  } ],
  "location": "--",
  "myuidtoshow": int,
  "name": "--",
  "nickname": "--",
  "phone": int,
  "picture": "--",
  "uid": "auth0|int",
  "user_id": "auth0|int",
  "username": "--"
}

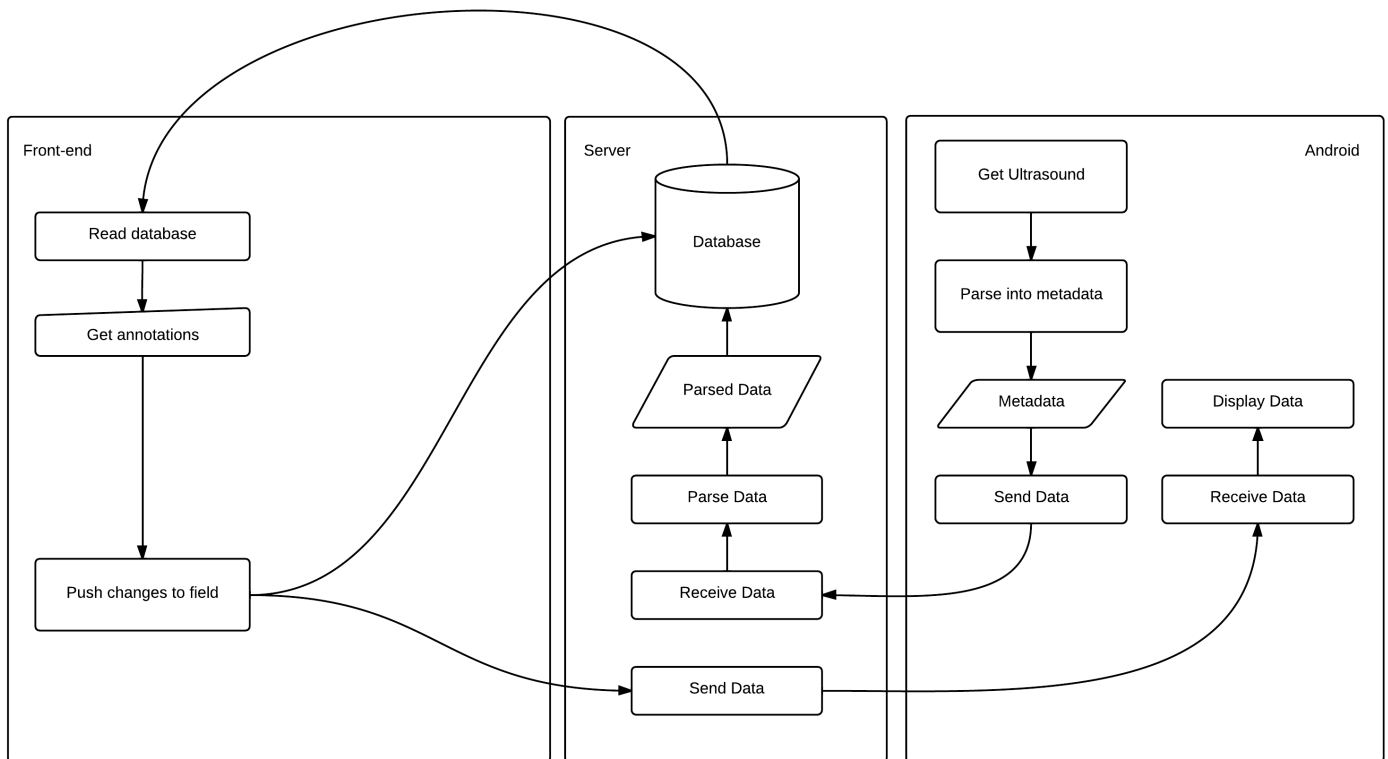
```

From this information, we can uniquely determine every user using our application. One thing to notice is the authlevel parameter. This is where we store the user's privileges. Depending on the user's privileges, they are able to do certain things within our application. For example, create new users, delete users and tasks, etc.

We have also included a number of third-party libraries with our front-end application.

- Windows Azure SDK – For access to Azure-based services
- Apache Commons Codec – For decoding Auth0 base64-encoded json responses
- Google Gson – For parsing json responses from Auth0
- Sqljdbc – For connection to SQL Server based databases
- JGoodies FormLayout – Library for building enhanced form layouts
- MiG Layout – Graphical Interface Layout designer

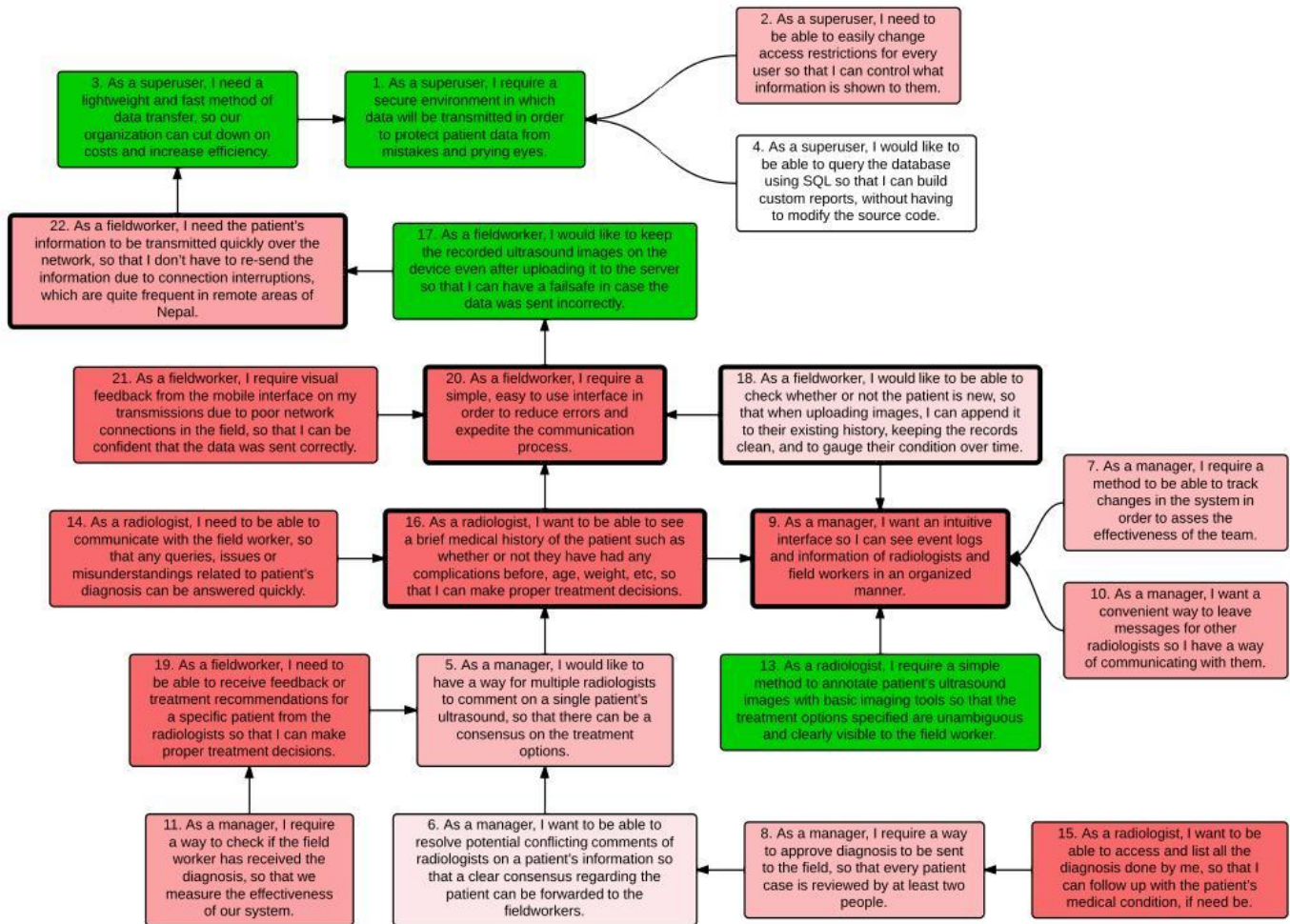
The interchange between the two applications can be seen in the following diagram (ignoring authentication):



The client software works directly on the database, except for when an image needs to be uploaded or downloaded. In the Records table, the images are pointed to by a link. (We were unable to insert an image directly into the database as Azure Mobile Services would stop seeing the table if we did so) This link is to where the file resides on the server. To retrieve or upload a file to the database, we connect through an endpoint on the server, and issue either an HTTP request, or post command. The images are stored in the PNG format, as they are lossless, as well as compressed by default. This compression can save a large amount of bandwidth when sending images overseas. This is one of our constraints, as prolonged network connectivity is an issue in remote locations. Once the data has been saved into the server, and the link established, the front-end queries the database for updates. If an update is found, it pulls the metadata from the database, and presents it to the user. If a user is interested in working on the request, the relevant image is then pulled from the server. Once the image has been annotated by the user, the annotations (without the original image, as we will overlay the two on the mobile device) are sent back to the database, as well as any addition metadata that was added, before flipping a completed flag in the database. When the Android client sees this flag, it will pull the annotations from the database, as well as any additional metadata added by the client user, overlay the annotations overtop the original image, which was saved in internal memory, and display the new metadata.

To help us determine the order in which we should implement our technologies, and to see the whole picture, we created a dependency graph of our user stories, where every block is color-coded to it's estimated size. White is smallest, red is largest, and green is completed. When choosing our technologies and frameworks, we referenced this chart in order to keep a realistic perspective on the relative importance of that task. We made the decision early on, that if we

could find something that was already built, instead of building out own components, such as the authentication system, it would allow us to allot more time to things such as the UI, and the communication between the database and the clients.



Technologies Used:

Android Application:

Programming Language: Java

IDE: Eclipse

Libraries: Android API (SDK), Auth0, Windows Azure Mobile Services

Android OS MIN Version Required: 3.0

Database: MSSQL hosted on Windows Azure

Website: PHP 4.4 hosted on Windows Azure Server

Framework for web development: CodeIgniter

Standalone Java Client:

Libraries: See page 2

IDE: Eclipse

Language: Java

## Android Client Design Details:

### Modules:

AuthenticationActivity.java: First thing to load to allow user to login.

Login Screen.java: Secondary Login to Launch Interface

WalkthroughScreen.java: Shows Tutorial video to user

WelcomeScreen.java: Loads Main Menu System in android device.

UltrasoundImageScreen.java: Loads Camera application in android to click picture.

MetadataScreen.java: Allows user to enter details about patient and upload them.

PatientRecords.java: Displays a list of patients added by the fieldworker.

PatientData.java: Displays a profile of specific patient.

Search.java: Displays search interface on mobile device to search a patient and calls PatientData.java to search

ImageDisplay.java: Displays Ultrasound Image downloaded from database on mobile device screen.

- All Code Files call functions provided by Android API (library).
- Android Client Follows Model View Controller Design throughout the code implementation.
- XML files are displayed as views and activity classes act as controllers monitoring user input and manipulating the view. Patient data involved in the app acts as model.

### Features that we would like to include in future:

- Allow android app to take multiple pictures of patient ultrasound from different angles and then send all those pictures tagged with one record.

## Java Client Bugs & Weaknesses

- Storing the Auth0 client ID and client secret in Secrets.java is a potential security risk, as decompiling the application would expose this. It would be best to protect this file by encrypting it, and storing it into a read-only, hidden system file.
- Storing the database connection string in Transmission.java is a potential security risk, as decompiling the application would expose this. It would be best to protect this file by encrypting it, and storing it into a read-only, hidden system file.
- Communicating directly with the database is a potential security risk. It should be moved server-side, where encrypted JSON tokens are exchanged.

## Server/SQL Bugs & Weaknesses

- Patient details are not encrypted.
- There is no access control when downloading or uploading images to the server. We could pass the user's Auth0 JSON token for authentication, along with the HTTP request, as the token is signed. (See: JSON Web Token)

## Android Bugs & Weaknesses

- Patient Records are not updated automatically.