Design and Technology Manifesto

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 three services from Windows Azure:

- SQL Database
- Cloud Storage
- Mobile Service

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 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:


```
{   "clientID": "aB3HL4XZm4J1Xc6CURHVCqtv2OgTgDjG",
    "created_at": "2013-10-20T02:54:43.322Z",
    "email": "assylay.sainova@mail.utoronto.ca",
    "email_verified": true,
    "identities": [
      {
          "user_id": "526345f3161ab8a17c00002c",
          "provider": "auth0",
          "connection": "Username-Password-Authentication",
```

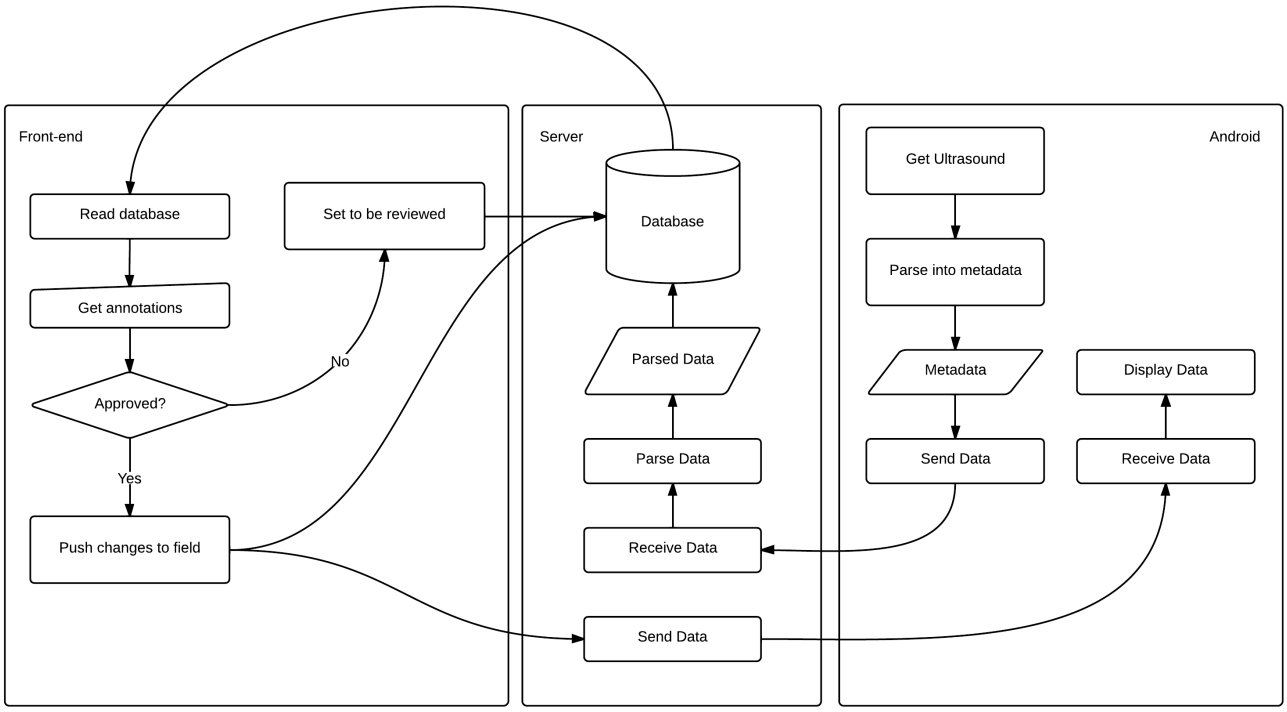
```
        "isSocial": false
      }
    ],
    "name": "assylay.sainova@mail.utoronto.ca",
    "nickname": "assylay.sainova",
    "picture": "",
    "user_id": "auth0|526345f3161ab8a17c00002c",
    "phone": "1234567890",
    "location": "Toronto",
    "firstname": "Assylay",
    "lastname": "Sainova",
    "authlevel": "1"
}
```


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. Many of these entries are added automatically, with more that we have added on when we created the user through a custom user-creation script.

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
- 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 server side is completely taken care of in the database itself. The sent images from the Android client are compressed using gzip before transmission. By doing so, we 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 database, the front-end queries it 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 image is then pulled from the database, and decompressed. Once the image has been annotated by the user, the annotations (without the original image) 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 picture more broadly, 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 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.