**ELEVATOR PROJECT**

**PACE System Hand Over**

**Application description:**

Manages records for potential dementia patients. Clinicians sign in to the system as a member of a clinic, can then add, update and remove patients from the system. All the information can be shared between the users of the same clinic and users can share data directly with others from different clinics. The records are separated into two varieties, private identifying values and cloud-suitable values. The private values are shared directly between users over P2P using WebRTC and are stored on the client using IndexedDB. Users are placed on a peer network. The structure of the network, the maintenance and the communication is all managed on the cloud. The Google App Engine is used to house the cloud-based stuff. An identifier is generated once the patient is created using their name and date of birth and acts as the primary key for the database on the cloud and the distributed database shared by the peers. This key is then used to combine both halves of the records into one.

**URL:** cloud-clinic.appspot.com -> The appspot.com address is for all App Engine apps that don’t have their own domain

**Dev Account Details:**

Address: [elevator.dcu@gmail.com](mailto:elevator.dcu@gmail.com)

Password: E713iZHy7R

This was used as the main developer account for the project as it needed to be a Gmail account and not a Gmail Apps account (DCU mail). Log in with this onto <https://console.developers.google.com/project/cloud-clinic> to see the basic app details, logs, DB stuff and administrative controls.

**User Account Details:**

To save the hassle of making accounts for every user to goes on and spending too long making a full user access control module, the system in place for users is to sign in with a gmail account. They’re then presented with a list of possible clinics and they must then enter the password specific to the clinic.

Right now there is only one clinic, DCU, which is added by default in code in the AuthenticateServlet.java file. The password for DCU is wDPZ5h40.

**Technologies Used:**

IndexedDB – Used to store all the patient private records on client. IDB.js has all the implementation stuff for working with it.

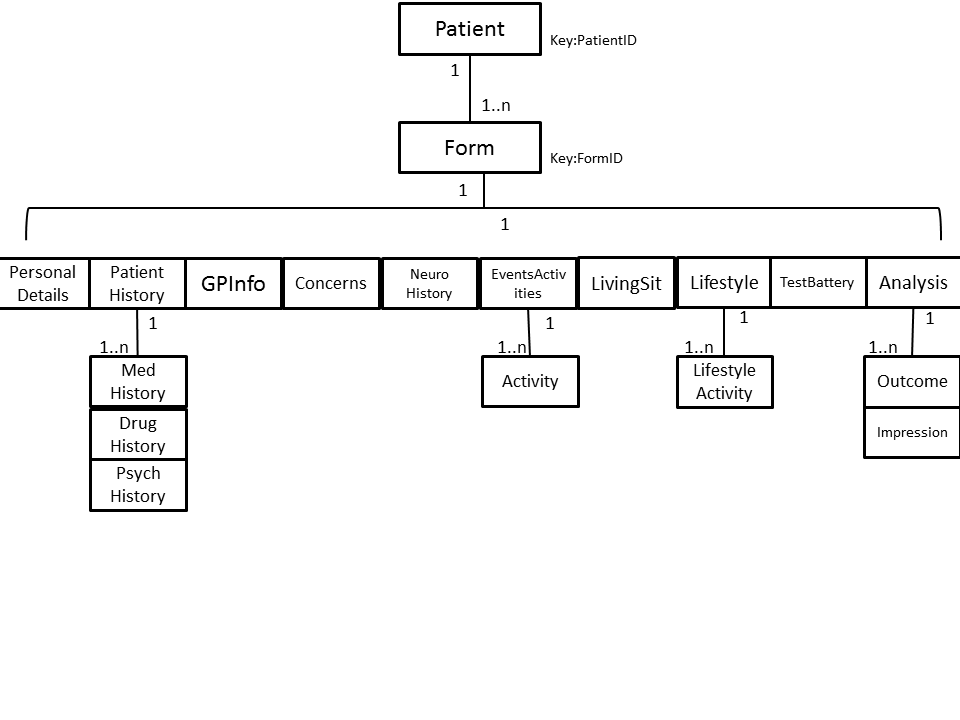
WebRTC­ – This is used to allow P2P communications to be set up. Requires the users get their real network via an ICE server. It has been implemented using the PeerJS library and uses their servers to get a peer’s address and to facilitate the communications. Right now, it only works properly with Chrome. Firefox can create peers successfully but it can’t seem to form a connection just yet and IE doesn’t support WebRTC.

Google App Engine – Used for all the cloud stuff. Built the application as a typical Java Servlet application. The Google Datastore is their own NoSQL method of storing information, contains all the P2P information necessary for the network. Cloud SQL is the SQL database used to store patient records. The information required to connect to the data is in the file: hibernate.cfg.xml. There are two sets of properties in this file, one is for the production version of the database which is on the cloud and the other is for development purposes. One set should always be commented out. Apache’s Common BeanUtils library is used to transform the form input from a page on the questionnaire into a Java object representing that page. However, some values are not always correctly stored and so these are done manually. GSON is then used to package objects retrieved from the database to be sent to the user.

Each page of the questionnaire has a matching servlet. Then there are a few utility servlets. The FileDispatchServlet is used to help redirect to another JSP file. The Authenticate servlet is used at first contact to check whether the person trying to sign in is a registered user. The ClinicOnTheCloud servlet is used to manage administrative duties such as logging users in, creating new clinicians and clinics, managing requests on data and facilitating communication.

**Database Structure:**

For the database, there are multiple patients. Each patient can contain multiple forms, corresponding to each time they visit the clinic.



**Protocol for Changing the Questionnaire:**

As data progresses from the client application to the server, on to the database and then back to a client, there are multiple areas that need to be looked at when updating the questionnaire with any new values or anything.

1. The data should be changed on the relevant html file for that page of the questionnaire.
2. The data object matching the changed page needs to be updated with the new values. The variable name should match the value of the ‘name’ attribute in the html. These variables also need to be annotated with ‘Expose’. This is so when the object is packaged by GSON, the variable is included. Without the annotation, the variable will not turn on client side. The getters and setters for each variable also need to be added.
3. The database should also be altered to include these variables. This usually must be done manually through MySQL workbench. Again, the column names must match the variable name or Hibernate won’t know how to match them (without the use of a special annotation).
4. The review jsp pages must then be updated with the new variables. Names must match.

**P2P**

As mentioned, WebRTC is used to manage the P2P communications and this is done with the PeerJS library: http://peerjs.com/. We’re currently using their free servers for setting up the connections which can support up to 100 connections at the same time. They also provide the server code that could be deployed somewhere to disassociate ourselves from a third party server. To use their server you need to include a dev key tied to the account which is 52wkj9kt0t9ms4i. To create a peer, use the dev key, and the urls for an ICE server and TURN server. These two servers are used to discover the client’s real network address which is needed for WebRTC communications. Once this is done, the cloud is informed of whether the client has WebRTC capabilities. If not the client uses Google Channel API and AJAX calls to send and receive the patient information.

When sending patient information over WebRTC, there seems to be a problem with communication passed the initial contact. To circumvent this, there is ony one point of communication between the peers and that communication contains the patient information in JSON format under the metadata tag. The rest of the information should be fairly straightforward from the Comm.js file which manages all the P2P communications.

**Possible Work To Be Done:**

Kate Irving asked me to do the following but I ran out of time:

Red Flags: Under certain circumstances, on the analysis page, the clinician should be alerted to values or answers that could influence their diagnosis/treatment. This includes medical results beyond the normal amount, any time a family member/friend provides information through the collateral forms, bad test results, large dips in activities, concerns or neurological-symptoms that have gotten worse or are very bad.

New Test: Kate also mentioned putting a new test in the test battery page. This would involve a bit more work calculating the score as answers are dependent on each other. Will need to talk about implementing this with a nurse to figure out how it should work.

Personal Notes on future improvement

Form Redesign: Although never mentioned by the nurses, it might be handy to update the patient questionnaire to appear more like the review screen. Then the clinician can go back and change information without possibly losing all their old information.

WebRTC: In the future, if the plan is still to use P2P to share confidential information, I would recommend implementing the P2P communications with WebRTC by itself and without PeerJS. Using a third part library adds a dependency on something we don’t have control of.

**Project Dependencies:**

GSON:

Gson-2.2.4-javadoc.jar

Gson-2.2.4-soures.jar

Gson-2.2.4.jar

Apache Commons:

Commons-beanutils-1.9.1 – All associated jar files

Commons-collections-3.2.1 – All relevant Jar files

Commons-logging-1.1.3 – All relevant jar files

App Engine SDK:

App Engine 1.9.9 Jar files. Should just be downloaded with the App Engine plug in

Hibernate:

Hibernate jar files

Antlr-2.7.7.jar

Dom4j-1.6.1.jar

Javassist-3.18.1-GA.jar

Jboss-logging-3.1.0.GA.jar

Jboss-transaction-api\_1.1\_spec-1.0.1.Final.jar

Mysql-connector-jave-5.1.30-bin.jar

**Note on client debugging**

There is a small black rectangle in the very top right hand corner of the home page. When clicked it opens up a textarea with various debugging information. To remove, in the home.jsp file delete the div with the id ‘infotext’. In order to write things to the debug area, use the following format: ‘$("#infotext").append("<div>"+Debug Text+"</div>"); ‘.