**Circle of Trust configuration**

Firebase:

We are using Google sign-in for authentication purposes. The server is currently using a personal Firebase account.

1. The authentication key from the account must be given in the Android Java code. Ensure to give the *Android key* to the Android code. This code can be generated by going to [https://console.cloud.google.com/apis/credentials?project=\*\*PROJECT](https://console.cloud.google.com/apis/credentials?project=**PROJECT)\_NAME\*\* and generating a new key for Android.
2. Another important note is to add your Android Studio’s SHA-1 key to the firebase project settings under “SHA certificate fingerprints”. The SHA-1 key can be found by going to gradle tasks in your Android Studio, and running “signingReport”.
3. Additionally, the web application client ID, found at the same above URL under “OAuth 2.0 Client IDs”, must be given to gtoken-signin.php. Once these steps are taken, users will be able to sign in and their account will be authenticated with our firebase application, granting the app access to our server.

In order to send push notifications to the phones, we are using Firebase Cloud Messaging. The server uses a server key which can be found under the settings -> cloud messaging tab of the firebase project. The url should be similar to <https://console.firebase.google.com/project/circleoftrust-8c17b/settings/cloudmessaging/android:edu.ucf.CD9>. The developers can use this key with all of the push notification requests.

AWS:

The server is currently running on an AWS UCF IT research server. Inside of our cloud we have a private and public subnet. The public subnet houses the API endpoints and a Bastion host for the developers to reach the database directly. The private subnet has a MariaDB database on an RDS instance.

The server can connect to the database using the connectDB.php endpoint. This endpoint connects to the RDS instance and signs in using credentials that we have stored. The credentials are currently being stored in a php file that is not available to the web. This was done by storing the file outside of the html folder, particularly in the www folder. The connectDB.php calls the credentials php to set the credential variables, and then is able to connect to the database.

The server also uses Rekognition and Comprehend so keys for these services must be provided in Analyze\_text\_on\_demand.php.

As a side note, we were in the process of making all of the API keys be handled like the database credentials when we turned the project in. Storing all the keys like this would result in better security and also an easier way of being able to change all the keys at once. In order to complete this process, one would just have to find all the plaintext keys in the endpoints, and move them to a new php file outside of the html folder. Then, whenever an API needs to be called, simply “include” this credentials php file.

Azure:

Inside of Analyze\_texts\_on\_demand.php, you will find the call for Azure’s content moderator. Content Moderator gives us scores based on three categories, and this is used for our flagging solution. This is currently set up using a free trial on a personal account. Information about this service can be found in the comments, and the flagging thresholds can be modified on line 339.

Cloud9:

On our AWS account, we have been using a Cloud9 instance called “CoT\_Server” to make changes to the server and view the server error logs. We also have a Cloud9 instance, “CoT\_Bastion” for the Bastion host. This instance must be running in order to view the database. Once the Bastion is running, navigate to its EC2 instance and go to its IPv4 address to gain access to the database. Make sure to include “/phpMyAdmin” after the IPv4 address.