**3.1 Setting up PlayChat using Firebase and AppEngine**

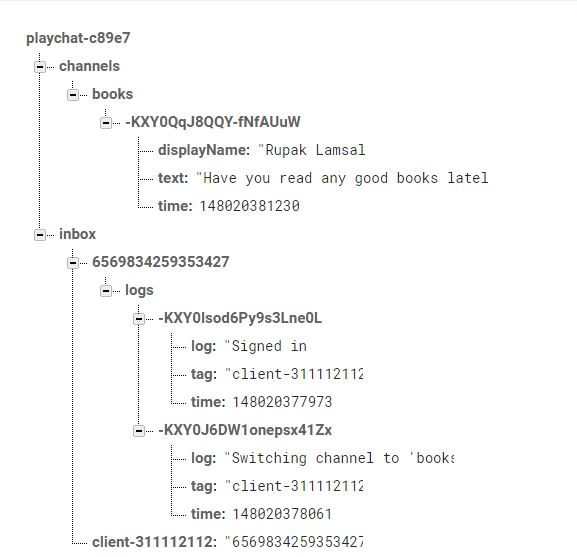
**c)**

Explaining the data stored in Firebase Database:

Tasks noted:

1. Signed in using google account
2. Switched to books channel
3. Entered a message – “Have you read any good books lately?”

Firebase database:



Explanation:

At the bottom, inside inbox, we see a randomly generated key that represents user’s account login, in this example: client-311112112. The currently listening servlet (16-digit numeric identifier -6569834259353427) to log events from the user lies next to it. Right above it, still inside inbox, is the same servlet identifier that is collecting logs. Under the /inbox/6569834259353427 are the user logs written by the app to that servlet. Above this, under channels, we can see the books channel that I switched to while performing my task. It contains the message that I wrote inside this channel. With every single event, there is a tag that refers to the servlet that was tracking it, and a timestamp that shows the order it was performed in.

**3.2 Saving Servlet Data into Cloud Storage**

**a)**

How servlets are saving client data currently?

Servlets are saving client data in the initLogger() method inside MessageProcessorServlet.java. When a servlet instance first starts, init(ServeltConfig config) is called where the servlet connects to the Firebase Realtime Fatabase and adds a listener to the /requestLogger/ data location. So, when a new entry is added to this location, servlet uses Firebase transactions to ensure only one servlet is listening at a time and accepts the work to process logs for the user. After that, initLogger() method is called where the servlet responds by retrieving the new log data. By using Message purger, it registers to the log branch and adds a log entry its concurrent linked queue of logs when a new child is added to that branch. This log is then later simply displayed in the doPost() method.

**b)**

**How Google Storage works?**

Explaining the steps needed to take to allow servlets to save data into Google Storage.

Setting up Google Cloud Storage:

1. First, you’ll need to set up the environment project first, then
2. Activate a cloud storage bucket. A default cloud storage bucket named with <project-id>.appspot.com will be created. Default permissions will allows us to read and write.
3. Then, download the client library using maven
4. And finally, use the client lib with the dev app server

To read and write to Google Cloud Storage, following changes will be needed to be made in the back-end code:

1. First, import required classes
2. Specify the cloud storage bucket you want to save your file to
3. Writing to cloud storage should be done inside doPost() method
4. Using the bucket name and a file name, created a GcsFilename
5. Created a default instance of GcsFileOptions
6. Using GcsFilename, GcsFileOptions and GcsService, you’ll be able to create a GcsOutputChannel where you can store the logs.

**3.3 Distributing Workload to Servlets**

**a)**

Currently, servlets are being distributed based on first come first serve. Using transactions, only one servlet can handle a request at a time, because a transaction can end in two ways only, with a commit or with a rollback. When assingning randomly generated inbox value to mutable current data, transaction’s doTransaction() method either runs, modifies code and saves, or fails, rolls back, and undoes the effects made (if any). This ensures that not more that one value is saved in the currentData, allowing only one servlet (the first one) to work at a time.

**b)**

**Distributing workload to servlets**

To custom distribute incoming client requests to servlets based on the metric of average requests of seconds, you’ll need to register a ServletContextListener in your web.xml file as follows:

<listener>  
  <listener-class>com.company.MyListener</listener-class>  
</listener>

Then, you will need to supply a class alongside your servlet and filter the code. For instance:

public class MyListener implements ServletContextListener {  
  public void contextInitialized(ServletContextEvent event) {  
    // This will be invoked as part of a warmup request, or  
    // the first user request if no warmup request was invoked.  
  }  
  public void contextDestroyed(ServletContextEvent event) {  
    // App Engine does not currently invoke this method.  
  }  
}

By using a ServletContextListener, you can run custom logic before any of your servlet is first invoked upon request to a new instance.