AD Project Individual Reflection

By Khan Sher Mostafa Somik (A0249468L)

Roles and Responsibilities

- Screen wire-frame drawing (rough)
- Entity/domain relationship and attribute planning
- Project sprint planning
- Daily standup meetings to assign task and follow up on completed tasks
- Android Activities (design & code)
  - MainActivity for chat groups
  - ProfileActivity
  - LoginActivity
  - RegistrationActivity
  - Bugfixes
- Android backend utility classes
  - For calling API using raw HTTP commands
  - For wrapping API calling class for easy implementation in Activities
  - For reading files from android's gallery/photos/files manager without accessing file-system directly
  - For uploading said files from android to java
  - Integration with Google's keyboard Gboard to receive GIF/Stickers
  - Bug-fixes
- Java backend utility classes
  - For calling python API using raw HTTP commands
  - For receiving uploaded files and saving to disk and providing a link
  - To read and output files when user calls on the link
  - Bug-fixes

Major Challenges & Lesson Learned

- **Technical aspects**
  - Implementing a Android Java class usable by anyone to call java back-end API
    - Using raw HTTP commands to "talk" to java back-end was difficult as I had to learn how HTTP commands works.
    - After going through over a dozen different codes, I decided to write my own class from scratch.
    - The class needed to be able to handle GET, POST, PUT, and DELETE requests.
    - The class needed to be able to upload files using multi-part form POST request.

```
// Post file by android FILE
public String postFile(InputStream fileStream, String fileName) {
    String crlf = "\r\n";
    String twoHyphens = "--";
    String boundary = "*****";

    try {
        conn.setRequestMethod("POST");
        if (token != null)
            conn.addRequestProperty("Authorization", token);
        conn.setDoInput(true);
        conn.setDoOutput(true);

        conn.setRequestProperty("Content-Type", "multipart/form-
data;boundary=" + boundary);

        // Open connection to server
        OutputStream os = conn.getOutputStream();
        BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(os));

        // Include the section to describe the file
        bw.write(twoHyphens + boundary + crlf);
        bw.write("Content-Disposition: form-data; name=\"file\";
filename=\"" + fileName + "\"" + crlf);
        bw.write("Content-Transfer-Encoding: binary" + crlf);
        bw.write(crlf);

        bw.flush();

        // Send the actual file contents to server
        InputStream fis = fileStream;

        int bytesRead;
        byte[] buffer = new byte[1024];
        while ((bytesRead = fis.read(buffer)) != -1) {
            os.write(buffer, 0, bytesRead);
        }
        os.flush();

        bw.write(crlf + twoHyphens + boundary + twoHyphens + crlf);
        bw.flush();

        // Close streams
        os.close();
        bw.close();

        int responseCode = conn.getResponseCode();
        System.out.println("Response Code :: " + responseCode);
        if (responseCode == HttpURLConnection.HTTP_OK) { //
connection ok
            return getData(); // Get json reply from server
        }
    } catch (IOException e) {
        System.out.println(e);
    }
    return null;
}
```

- The class needs to be able to send properly formatted authorization token in the header of every request.

- The class needs to have another wrapper class that hides the raw calls to make it more user friendly.
- The wrapper class needed to be versatile enough to handle multiple types of input format and always reply with the proper Data Transfer Object (DTO), a java class that is used to access the data provided by the java back-end server.

```
  public <T> ArrayList<T> jsonToArrayList(Class<T> clazz) {
         if (TextUtils.isEmpty(result)) {
             return null;
         }
         Type type = new TypeToken<ArrayList<JsonObject>>() {
         }.getType();
         ArrayList<JsonObject> jsonObjects = new
Gson().fromJson(result, type);

         ArrayList<T> arrayList = new ArrayList<>();

         for (JsonObject jsonObject : jsonObjects) {
             arrayList.add(new Gson().fromJson(jsonObject, clazz));
         }
         return arrayList;
     }
```

- The wrapper class should be able to run the API calling class in a background thread and have some mechanism to sync data between the threads upon reply from java back-end server.
- One of the lessons learnt is that it is not good to make the main thread “wait” for result as this makes the app seem to hang. This is ok for activities like login, but not for activities like file upload, which may take some time to upload.

```
new Thread(() -> {
    int replyTo = -1; // Disabled
    if (message.trim().length() == 0) return;

    APICommunication apiComm = new APICommunication(token);
    apiComm.sendMessage(message, userId, groupId, replyTo);
    String reply = apiComm.getResult();

    if (reply.equals("Message Saved")) {

        runOnUiThread(() -> {
            txtChatMsg.setText("");

            getChats();
            if (mMessageAdapter == null) {
                initChatList();
            } else {
                mMessageAdapter.setMessageList(messages);
                mMessageAdapter.notifyDataSetChanged();
            }
            scrollToBottom();
        });
    }
}).start();
```

- Opening files for upload on Android 10+
    - As android 10 removed support for public folder access for apps without some big workarounds, it was very difficult to figure out an alternative way to access the files without requesting such permissions.
    - The best workaround was to use existing apps to “pass” the file to our app.
    - Our app creates an intent requesting for androids default or built in gallery/photos/filemanager to hand over the file with specified type (image/video/everything).
    - Once the app replies, the new intent is used to read the file through the other app, negating the need for read/write access to filesystem.

```
// this function is triggered when user
// selects the image from the imageChooser
public void onActivityResult(int requestCode, int resultCode, Intent
data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (resultCode == RESULT_OK) {
        if (requestCode == SELECT_PICTURE) {
            // Get the url of the image from data
            Uri selectedImageUri = data.getData();

            if (selectedImageUri != null) {

                Toast.makeText(this, "Uploading image in
progress...", Toast.LENGTH_LONG).show();

                new Thread(() -> {

                    String fileName = "upload." +
getMimeType(selectedImageUri);

                    InputStream inputStream;
                    try {

                        inputStream =
getContentResolver().openInputStream(selectedImageUri);

                        APICommunication apiComm = new
APICommunication(token);
                        apiComm.uploadFile(inputStream, fileName);
                        String result = apiComm.getResult();

                        if (result != null && result.length() > 10) {
                            Log.d(">> RESULT ::", result);
                            FileDTO fileDTO =
apiComm.jsonToObj(FileDTO.class);

                            Map<String, Object> postDataMap = new
HashMap<>();
                            postDataMap.put("id", userId);
                            postDataMap.put("photo",
fileDTO.getUrl());

                            apiComm.updateUserPhoto(postDataMap);

                            runOnUiThread(this::showProfileData);
                        }
                    } catch (Exception e) {
                        Log.d(logTag, "File not found: " + e);
                    }
                }).start();
            }
        }
    }
}
```

- Implementing a Java class to interact with Google's keyboard Gboard to receive Gif/stickers sent by the keyboard
  - This was done to help the user react with a gif/sticker without having to find it on the phone/interent and upload to server.
  - The base code overwrites Android's built in EditText and creates required overrides to accept media content from Gboard keyboard.
  - Parts of the code was written by my group mate but she could not get the "onCreateInputConnection" to work as she did not know how to receive the file from Gboard and display it.
  - The main issue was lack of documentation so it was mostly a done by decoding the object "inputContentInfo" to see how the data was being passed on by Gboard and with a lot of trial and errors.

```
@Override
public InputConnection onCreateInputConnection(EditorInfo editorInfo)
{

    final InputConnection ic =
super.onCreateInputConnection(editorInfo);
    EditorInfoCompat.setContentMimeTypes(editorInfo, new
String[]{"image/gif", "image/jpg", "image/jpeg", "image/png"});

    final InputConnectionCompat.OnCommitContentListener callback =
            (inputContentInfo, flags, opts) -> {
                // read and display inputContentInfo asynchronously
                if (BuildCompat.isAtLeastNMR1() && (flags &

InputConnectionCompat.INPUT_CONTENT_GRANT_READ_URI_PERMISSION) != 0)
{
                    try {
                        inputContentInfo.requestPermission();
                    } catch (Exception e) {
                        return false; // return false if failed
                    }
                }
                String message = "[img=" +
inputContentInfo.getLinkUri().toString() + "]";
                String mime =
inputContentInfo.getDescription().getMimeType(0);
                this.setText(message);
                imgSelectCallback.onImageSelected(message, mime);
                return true; // return true if succeeded
            };
    return InputConnectionCompat.createWrapper(ic, editorInfo,
callback);
}
```

- Implementing Java a file upload handler and reply with uploaded file upon request
    - The Java code for SpringBoot file upload was spread over a few websites so a lot of trial and error went into this as well.
    - Main issue with this was the upload was working fine on local but was failing once the Java back-end app was built, packaged and deployed on cloud.
    - The reason was the multipart file upload was failing when it was passing through nginx reverse proxy that we used to provide SSL support for java back-end as well as react.
    - After I managed to get this part working, the second issue was Tomcat crashing when request to download the uploaded file was made.
    - Thus, i had to rewrite the whole FileController & FileStorageServiceImpl to not only upload files, but also reply with the content of the file upon request with proper headers to define file name and media type.

```
// View or Download files
@GetMapping("/{filename:.+}")
@ResponseBody
public ResponseEntity<?> getFile(@PathVariable String filename) {
    try {
        Resource file = storageService.load(filename);
        InputStream in = file.getInputStream();
        MediaType mediaType = null;

        String type = storageService.getMimeType(file);
        if (type != null)
            mediaType = MediaType.parseMediaType(type);


        if (mediaType == null) {
            System.out.println(file);
            // Download file as attachment
            mediaType = MediaType.parseMediaType("binary/octet-
stream");
            return ResponseEntity.ok()
                    .contentType(mediaType)
                    .header(HttpHeaders.CONTENT_DISPOSITION,
"attachment; filename=\"" + file.getFilename() + "\"")
                    .body(file);
        } else {
            // Display file in browser
            System.out.println("\n" + mediaType.toString() + "\n");
            return ResponseEntity.ok()
                    .contentType(mediaType)
                    .header(HttpHeaders.CONTENT_DISPOSITION, "inline;
filename=\"" + file.getFilename() + "\"")
                    .body(new InputStreamResource(in));
        }

    } catch (IOException e) {
        System.out.println(e);
    }
    return null;
}
```

- For some reason, Java is not able to properly get the media type from file header for some files, such as PDF and MP4. As such, I had to generate a set of of mimeType based on the file’s extension.

- **Non-technical aspects**
    - One of the issues we faced was when my entire day’s work was overwritten by one of my team mates by mistake. She pushed her code without pulling my changes first, causing github to override my codes. Luckily we managed to catch this as early as the next day and restored my codes from the previous commit.
    - Another issue was for the first week, we did not have any daily standup meeting, and it was very difficult to link up with others and see what others are doing. As such, beginning of second week, I enforced daily meeting. Once it started, we quickly understood the benefits of this and our work went very smoothly from there.
    - Final issue was with regards to working remotely, which caused us to keep our issues to ourselves and not “disturb” our team mates. As such, we decided to meet-up every day even if it’s for an hour, which resulted in a higher productivity compared to the days we didn’t.