PROJECT 6

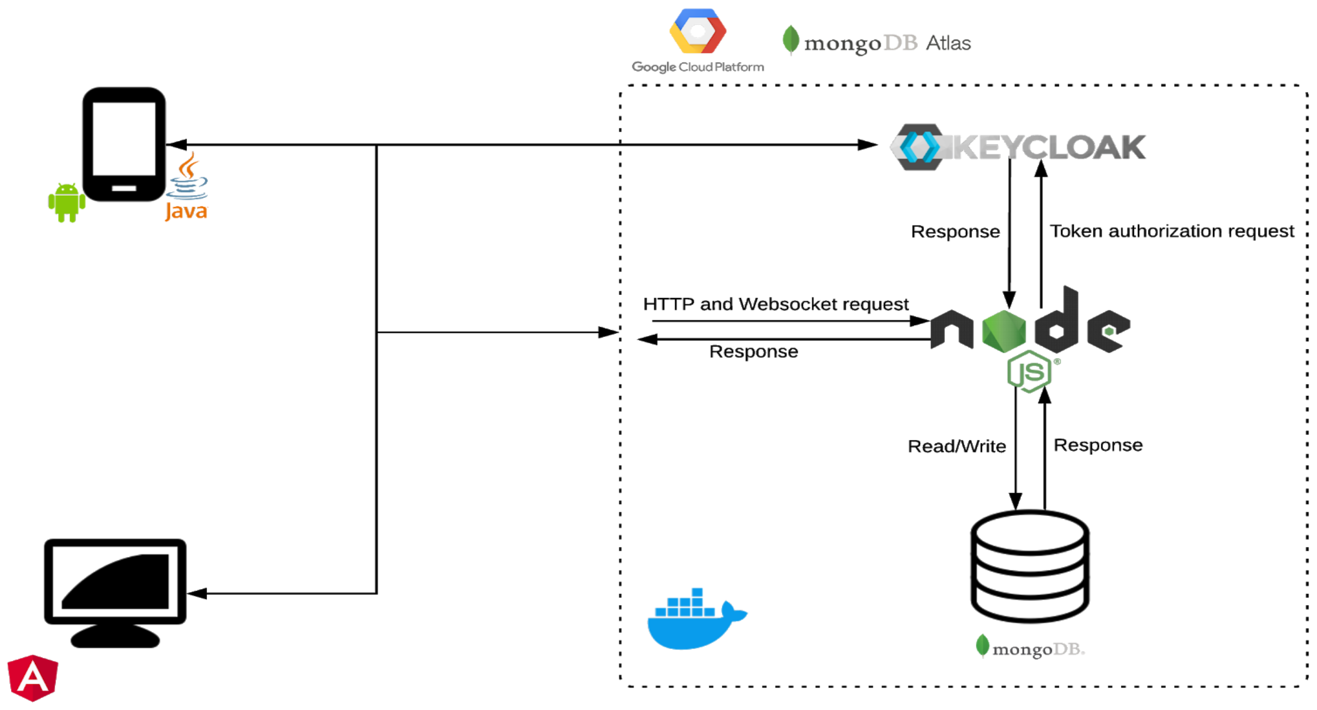
Project Summary:

- **Project Name**: Copy Once Paste Anywhere
- **Team**: Madhusudhan Aithal Mahabhaleshwara, Raj Chandak, Jitendra Marndi
- **Overview**: We have built a solution for synchronizing the clipboard content (any copied text, or URL) across all the devices a user has logged in. There are two types of applications for users - Android app and Web app. The user has to simply login to these devices in order to use our system. Whenever a user copies some text in any of their logged-in devices, it is copied on all of their other devices, ready to be pasted. This makes it much more convenient for users to transfer content from one device to another.

Final State of the System:

We have fully functional production-level Android and Web applications. These applications have been developed using Java and Angular respectively. The web application has been deployed on Google App Engine and is available here. The Android application can be installed using this APK. We have also written server code (deployed here) that enables decoupled communication between the Android application and the Web application. The GitHub repository links for all the 3 components of our system have been included at the end of this report.

- **Final Architecture Diagram**: Our final architecture diagram is as follows:



Following is a brief description of this final architecture:

- Angular has been used for developing the web application, and Java has been used for developing the Android application.
- Using the Socket.io library, we have used Websockets to maintain the connection between the user applications and the Node.js server. Whenever there is a new clipboard data, the Android app or desktop app send the data to the server and the corresponding controller in the server receives the message through this WebSocket.
- We have used WebSockets in our system because the data needs to be pushed to the server frequently and using request-response mechanism (RESTful web service) will have more overhead compared to WebSockets.
- We have used Keycloak as our authentication server that maintains user credentials and is responsible for issuing and invalidating auth tokens to clients (Android or web app). All the auth tokens are validated by this authentication server.
- We have used MongoDB ATLAS as our persistent data store.
- There are multiple controllers in the Node.js microservice that interact with the Keycloak authentication server and MongoDB.
- The node.js server, Keycloak authentication server, MongoDB database, and the Angular web app are containerized and deployed using Docker and Docker swarm. We have used Google Cloud Platform to deploy our infrastructure.

- **Goals Achieved**:

  - **The system allows users to copy and paste content across devices:** Users are able to copy on one device (eg. Android) and paste on any other devices (eg. Desktop/ Tablet), given that these devices have our compatible software running and the users have logged in to the system. For instance, when the user copies some text or URL on our Web app through his Windows laptop, that copied content is available to paste on his Android smartphone, given that the user is logged in to our software on both, the Web app and his Android smartphone.

  - **Users are able to copy any text or URL:** The system allows users to copy any piece of text or URL and makes it available to paste across any of their other devices onto which they have logged in.

  - **Users are able to see the synchronized clipboard history across devices:** Logged in users are able to see the entire history of what they have copied across devices, in the form of a clipboard. Whatever the users copy on their web app and mobile is visible and available to copy again or delete under the *Desktop* and *Mobile* tabs respectively.

- **Goals Not Achieved**:

  - **The system does not support sharing of media (images or videos):** We had initially planned on supporting media as well, however, supporting media content sharing was causing other issues related to sending, receiving and storing these data. Hence, we decided to not support this requirement in our application.

- **Functional Requirements Achieved:**

    - **The clipboard is synchronized across devices:** Whatever is copied in any one of the devices, is visible on the clipboard on all other devices, i.e. any compatible content that is copied on the desktop device (using the web app) should be available to paste on the Android device, and vice-versa.

    - **Users are able to sign in and sign out**: The user is able to sign in or sign out of the system, on both, the Web and Android version of the software.

    - **Users are able to see the clipboard history:** Whatever the user has copied is visible to him under the respective *Desktop* and *Mobile* tabs. Be default, the last 20 copied items are visible, when the user scrolls down, the system loads more items.

    - **Users are able to copy directly from the clipboard list:** Every item in the clipboard has a copy button and on tapping/clicking that button, that content is copied across devices.

    - **Users are able to delete any clip from either of the clipboards (Not initially planned, but we decided to add it):** The user can delete an item from the web app by simply clicking on the delete icon that is present in every list item or by swiping left/right on the clip item in the Android device.

    - **Users are able to see the device model that shows where the copied clip came from (Not initially planned, but we decided to add it):** For instance, if the user is using a Oneplus 6 for coping any content, then the copied clip has a field that shows "Oneplus 6" as the device name, depicting the device from where the clip origintated.

- **Functional Requirements Not Achieved:**

    - **Users are not able to search for any previously copied content:** We had initially planned to implement the search feature on both the Web and the Android app. However, due to time constraints and everything going remote, we were not able to implement it.

    - **Desktop Application:** We had initially planned on developing a Desktop application instead of a web application using the Electron.js framework. However, during implementation, we ran into issues with integration of Keycloak with the Electron.js based desktop application. Hence, we switched to developing a Web application using Angular that can be used on any device with an internet connection.

- **Non-functional Requirements Achieved:**

    - **The synchronized copy happens in the order milliseconds:** Whatever content the user has copied in one of his devices, is available to paste in his other devices in the order of a few hundred milliseconds.

    - **The system has a field that separates out the clipboard history in the database for *Desktop* and *Mobile* platforms:** There is a field in each MongoDB document that identifies where each copy originated from, i.e. either Desktop or Mobile.

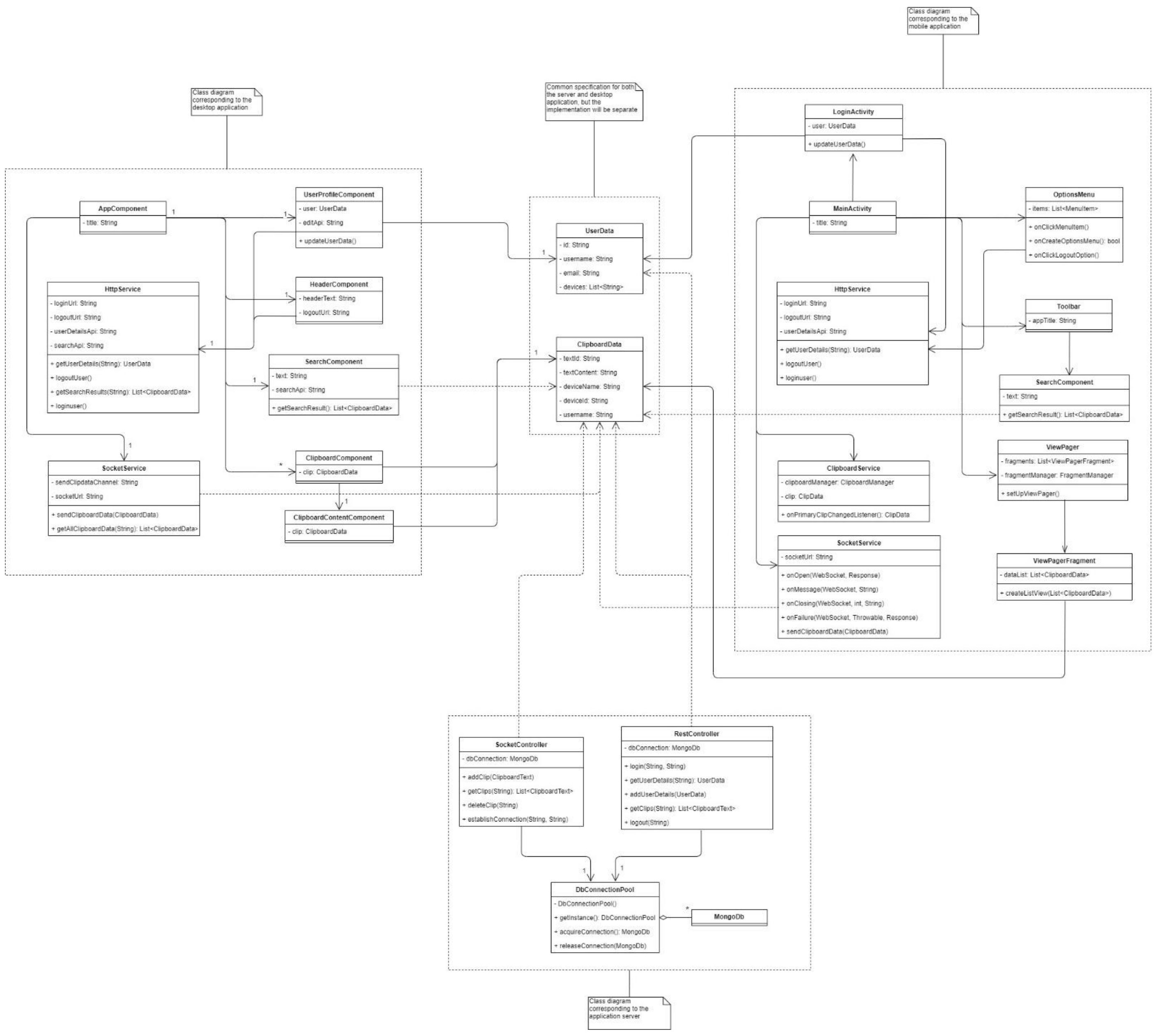
- **Clipboard data is persistent across user sessions:** No clipboard data should be lost if the user logs out, i.e. the copied content should not be dependent on the user session. Each copy should be registered in the database.

- **Multiple users DO NOT share the clipboard:** The system ensure that clipboards are not shared across users and any clip copied by UserA is not available to paste for UserB.

- **Non-functional Requirements Not Achieved:**
    - **Desktop clipboard should be independent of the Operating System:** As we were not able to develop the desktop application and instead switched to the web application, this non-functional requirement was not achieved.

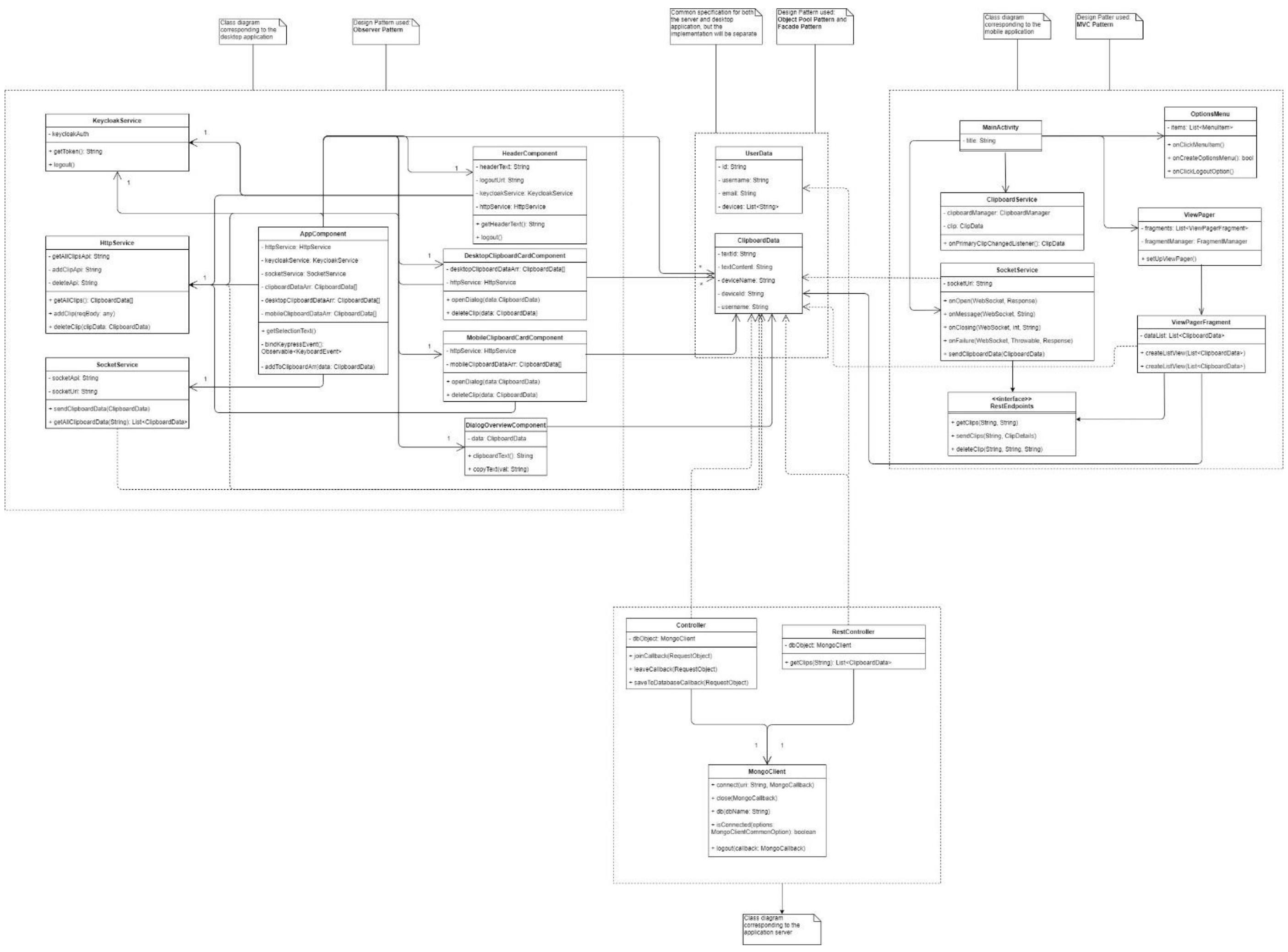
Final Class Diagram and Comparison:

- **Initial Class Diagram:** Following is the Class Diagram from Project 4:



Class diagram corresponding to the desktop application
Common specification for both the server and desktop application, but the implementation will be separate
Class diagram corresponding to the mobile application
Class diagram corresponding to the application server
AppComponent
UserProfileComponent
HeaderComponent
SearchComponent
ClipboardComponent
ClipboardContentComponent
HttpService
SocketService
UserData
ClipboardData
LoginActivity
MainActivity
OptionsMenu
Toolbar
ViewPager
ViewPagerFragment
ClipboardService
SocketController
RestController
DbConnectionPool
MongoDb

- **Final Class Diagram:** Following is our final Class Diagram:


Class diagram corresponding to the desktop application
Design Pattern used: Observer Pattern
Common specification for both the server and desktop application, but the implementation will be separate
Design Pattern used: Object Pool Pattern and Facade Pattern
Class diagram corresponding to the mobile application
Design Patter used: MVC Pattern
KeycloakService
- keycloakAuth
+ getToken(): String
+ logout()
HttpService
- getAllClipsApi: String
- addClipApi: String
- deleteApi: String
+ getAllClips(): ClipboardData[]
+ addClip(reqBody: any)
+ deleteClip(clipData: ClipboardData)
SocketService
- socketApi: String
- socketUrl: String
+ sendClipboardData(ClipboardData)
+ getAllClipboardData(String): List<ClipboardData>
AppComponent
- httpService: HttpService
- keycloakService: KeycloakService
- socketService: SocketService
- clipboardDataArr: ClipboardData[]
- desktopClipboardDataArr: ClipboardData[]
- mobileClipboardDataArr: ClipboardData[]
+ getSelectionText()
- bindKeypressEvent(): Observable<KeyboardEvent>
- addToClipboardArr(data: ClipboardData)
HeaderComponent
- headerText: String
- logoutUrl: String
- keycloakService: KeycloakService
- httpService: HttpService
+ getHeaderText(): String
+ logout()
DesktopClipboardCardComponent
- desktopClipboardDataArr: ClipboardData[]
- httpService: HttpService
+ openDialog(data:ClipboardData)
+ deleteClip(data: ClipboardData)
MobileClipboardCardComponent
- httpService: HttpService
- mobileClipboardDataArr: ClipboardData[]
+ openDialog(data:ClipboardData)
+ deleteClip(data: ClipboardData)
DialogOverviewComponent
- data: ClipboardData
+ clipboardText(): String
+ copyText(val: String)
UserData
- id: String
- username: String
- email: String
- devices: List<String>
ClipboardData
- textId: String
- textContent: String
- deviceName: String
- deviceId: String
- username: String
MainActivity
- title: String
OptionsMenu
- items: List<MenuItem>
+ onClickMenuItem()
+ onCreateOptionsMenu(): bool
+ onClickLogoutOption()
ClipboardService
- clipboardManager: ClipboardManager
- clip: ClipData
+ onPrimaryClipChangedListener(): ClipData
ViewPager
- fragments: List<ViewPagerFragment>
- fragmentManager: FragmentManager
+ setUpViewPager()
SocketService
- socketUrl: String
+ onOpen(WebSocket, Response)
+ onMessage(WebSocket, String)
+ onClosing(WebSocket, int, String)
+ onFailure(WebSocket, Throwable, Response)
+ sendClipboardData(ClipboardData)
ViewPagerFragment
- dataList: List<ClipboardData>
+ createListView(List<ClipboardData>)
+ createListView(List<ClipboardData>)
<<interface>> RestEndpoints
+ getClips(String, String)
+ sendClips(String, ClipDetails)
+ deleteClip(String, String, String)
Controller
- dbObject: MongoClient
+ joinCallback(RequestObject)
+ leaveCallback(RequestObject)
+ saveToDatabaseCallback(RequestObject)
RestController
- dbObject: MongoClient
+ getClips(String): List<ClipboardData>
MongoClient
+ connect(uri: String, MongoCallback)
+ close(MongoCallback)
+ db(dbName: String)
+ isConnected(options: MongoClientCommonOption): boolean
+ logout(callback: MongoCallback)
Class diagram corresponding to the application server
1
*

(If the details are not visible even after zooming in, please check this google drive link. Access it using your Colorado.edu account. It has all the images from Project 4-6.)

- **Comparison:** While we implemented almost all the features that we had initially planned, our class diagram did see some changes from our initially planned class diagram from Project 4. These comparisons are summarized in the table shown on the following page.

Class Diagram Component	Project 4 Class Diagram	Project 6 Class Diagram
Webapp (Previously Desktop)	**No Keycloak Class**: We had not included the Keycloak class as we were unfamiliar with Keycloak thought that it would probably not need a class of its own.	**Included Keycloak Class:** In our final class diagram, however, we have included the Keycloak class as a separate class in the Web application component.
Webapp (Previously Desktop)	**Search Component Class**: As we had initially planned on including the Search component, we had included this class in our initial class diagram.	**No Search Component Class**: However, as we were not able to implement this feature, we did not include it in our final class diagram.
Webapp (Previously Desktop)	**Clipboard Component**: Initially, we had not planned on creating separate components for Mobile and Desktop in the web application. However, to decouple our code as much as possible, we decided to make to separate components and hence got rid of the Clipboard component.	**MobileClipboardCard and DesktopClipboardCard Component**: Since we made separate components for the mobile and desktop tabs, we added these 2 classes to our final class diagram.
Webapp (Previously Desktop)	**No Dialog Overview Component**: We had not anticipated the situation when if a clip is too big to show appropriately. Thus, we did not think of any onClick function callbacks as we had no plans of making the individual clips clickable.	**Included Dialog Overview Component**: In order to accommodate for the situation mentioned on the left, we decided to make the individual clips clickable and showed the entire content of the clips in the Dialog box. The user is also able to copy the clipboard text from this dialog box.
Server	**DbConnectionPool Class**: Initially we had planned on using a different flavour of MongoDB which warranted the inclusion of the DBConnectionPool class in order to communicate with the 2 controllers.	**MongoClient**: However, with the use of MongoDB ATLAS, which is a cloud version of MongoDB, it necessitated the need of having MongoClient class with functions specific to MongoDB ATLAS.
Android App	**Search Component Class**: As we had initially planned on including the Search component, we had included this class in our initial class diagram.	**No Search Component Class**: However, as we were not able to implement this feature, we did not include it in our final class diagram.
Android App	**HTTP Service:** Initially we had planned on using the Volley library in Android for making REST calls. However, there was no appropriate documentation of using it with Keycloak. Thus, we decided to use the Retrofit library which has much better support with Keycloak.	**RestEndpoints**: This is an interface that declares all the endpoints that are used by the retrofit library to send or receive data from the server by sending the authorization Bearer token in the request header.

<u>Third-Part Code vs. Original Code</u>:

We will now go over each component separately to show what code was original vs. what code was derived from third-party sources.

Android Application:

As Raj was familiar with Android development and Material Design guidelines, all of the user interface code was original and written from scratch. This includes all the XML code and the Java code written for interacting with the User Interface. We did, however, take the help of Article [1] for add the swipe to delete functionality. This article explains and details out all the necessary design changes and callback functions required to implement the swipe to delete feature.

For using WebSockets, we used the official Socket.io documentation [2] for Native support for Android. The documentation was easy to read and the code snippets provided were sufficient to give us an idea of how to integrate Socket.io Websockets with our Android application.

For integrating KeyCloak with Android we used two third-party libraries, Retrofit[3] and AppAuth[4]. The token exchange flow described earlier is easily implemented using the classes provided by AppAuth. To learn the usage of AppAuth library we referred to its official API documentation[5], GitHub source code[6], and a codelab demo[7] by Google. Making RESTful API calls with the access token is implemented using Retrofit. It helps to provide a user-friendly interface for the RESTful server. To learn its usage we referred it official documentation[8] along with few YouTube tutorials[9]. To learn to secure our applications with KeyCloak we referred to KeyCloak official documentation[10].

While we referred multiple resources to learn the usages of libraries, we haven't taken any code as such from any of these sources. However, our own code logic is inspired by some of the tutorials and documentation. For example, for the token exchange flow, the AppAuth library hides the nitty-gritty of whole flow and provides us with a simple class such as *AuthorizationService* which can be used to call *performAuthorizationRequest* and the response from KeyCloak server can be stored in an *AuthState* object. Codes relating to *AuthorizationService* and *AuthState* are added in *MainActivity* and *ClipboardService* class respectively. The Retrofit library provides suitable annotations such as @GET and @POST which can be used to create an HTTP request. Interface *RestEndpoints* is created using those annotations. These HTTP requests serve as the RESTful API call.

[1] https://www.androidhive.info/2017/09/android-recyclerview-swipe-delete-undo-using-itemtouchhelper/

[2] https://socket.io/blog/native-socket-io-and-android/

[3] https://www.keycloak.org/

[4] https://appauth.io/

[5] https://openid.github.io/AppAuth-Android/docs/latest/

[6] https://github.com/openid/AppAuth-Android

[7] https://codelabs.developers.google.com/codelabs/appauth-android-codelab/index.html#0

[8] https://www.keycloak.org/docs/latest/securing_apps/index.html

[9] https://square.github.io/retrofit/

[10] https://www.youtube.com/watch?v=4JGvDUlfk7Y#action=share

Web (Angular) Application:

We looked at some examples online for making HTTP and socket requests in angular. Based on these examples, we wrote our own code for making HTTP requests such as GET, POST and DELETE, and also for emitting and listening for messages on a specific channel on socket. The HTML and class codes of all the components in the app were designed and implemented on our own. In our angular app we are using Angular Material for theming the application. We went through some examples and understood how to use certain elements of the Angular Material such as mat-buttons, mat-cards, mat-tab etc., and we implemented our UI designs using these code snippets after making the required changes based on our requirements.

We are using keycloak.js for integrating our Angular app with Keycloak. So we went through the documentation of keycloak.js and the below article for the integration. The article acted as a guide for us to successfully integrate keycloak with angular and we wrote the code on our own based on the information provided in the article.

We also went through the documentation of electron.js to see how to access clipboard service of the computer, how to make HTTP requests and how to establish the socket connection to the server. We used some of the standard code snippets available online for making HTTP requests and changed to fit our requirements. But we are not using this part of the codebase because of the Keycloak integration issue with electron.js.

[1] https://medium.com/@blained3/connecting-keycloak-to-angular-d175c92a0dd3

[2] https://material.angular.io/components/categories

Server-Side (Node.js) Code:

- We used the tutorials provided in the `getting started' section of the expressjs website mentioned below to build basic GET and POST APIs. With the help of this knowledge, we implemented the required REST endpoints on our own.
- We used the below tutorial of compose.com to learn how to use 'mongodb' Node.js' driver to establish connection to the MongoDB database.
- We also made use of the tutorial provided on the socket.io website (mentioned below) to see and understand how to use socket.io library in Node.js. The code for listening and emitting messages on different channels of the socket and accordingly sending the data is our own implementation.
- We went through a medium article to understand how to protect the REST APIs using keycloak. The keycloak-connect library which we are using for this purpose can be used in a straightforward manner, without many changes

[1] https://www.compose.com/articles/connection-pooling-with-mongodb/

[2] https://socket.io/docs/

[3] https://expressjs.com/

[4] https://medium.com/@ramandeep.singh.1983/enterprise-web-app-authentication-using-keycloak-and-node-js-c10b0e26b80d

---

Statement on the OOAD Process for the Entire Project:

Following are 3 key design and analysis issues that our team faced during the implementation of this project:

- **Desktop Application:** As mentioned before, we were not able to implement a working version of our desktop application. The issue we ran into was the integration of Keycloak with Electron.js. Electron internally uses a file system for navigation across screens. As Keycloak provides a basic login page which developers can integrate in their systems, we needed to send a request to a publically accessible website running on the HTTP protocol. Once the user authentication was done, Keycloak was responsible for navigating back to the Electron.js app and send an authState object. However, as Electron does not support navigation over HTTP directly, we were unable to resolve this issue. Hence, we developed a web application using Angular. The bane in disguise was that we were able to deploy this web application to Google App Engine, allowing users to interact with their clipboards on not just desktop devices, but any device having an internet connection.

- **Media (Images/Videos) Content Support:** Initially, we had planned on supporting media files in addition to plain text and URLs. However, supporting images and videos was resulting in many other problems revolving around sending, receiving and storing this data, possibly resulting in an explosive and unstable database. Moreover, supporting media content was resulting in significantly slower loads of the clipboard content. Hence, we decided to not allow for sharing of media files.

- **Copy Feature on Web App:** Since the Web app runs on a browser, it is difficult to track the copying of any piece of text from other applications. With the desktop app, this was possible as we were able to run a service in the background that tracked copying of text/URLs across any of the installed applications. However, due to security issues, we were unable to access the shared clipboard and hence not able to track clipboard copies on applications other than the browser. Thus, our webapp is able to track copies only happening in the browser itself.

Conclusion and GitHub Links:

Overall, this was really a fun project to implement. All the team members learnt a lot of new tools, frameworks, libraries and technologies that we were previously unfamiliar with.

We have created a GitHub Organization and added 3 separate repositories in this organization. All the TAs/Graders and the Professor have been given access to the entire organization. The link to the organization and the individual repositories are mentioned below:

OOAD Project GitHub Organization: https://github.com/OOAD-Semester-Project

- **Android App Repo:** https://github.com/OOAD-Semester-Project/android-app
- **Web App Repo:** https://github.com/OOAD-Semester-Project/copa-electronjs-app
- **Server Code Repo:** https://github.com/OOAD-Semester-Project/clipboard-server

OOAD Presentation Slides: Link

Web App: https://clipboard-sync-angular-app.appspot.com/

Server Link: https://clipboard-syncronization-app.appspot.com/