



Final thesis Degree
Computer Engineering Degree
Faculty of Mathematics and Computer Science
Universitat de Barcelona

**eParliament: Developing an Android solution
to manage a parliament session workflow**

Lucila Miriam Bermúdez Cuellar

Director: Néstor Tejero

Made with: Scytl

Barcelona, 1 of February of 2017

Index

List of figures.....	4
Glossary.....	5
1. Introduction	6
2. Objectives	8
3. Basic Parliamentary Concepts.....	9
4. Basic Android concepts	10
4.1. Activity	10
4.1.1. Life Cycle	11
4.2. Fragment.....	13
4.2.1. Lifecycle.....	13
4.3. Service.....	14
4.3.1. Lifecycle.....	14
4.4. Shared preferences.....	15
5. Basic Security concepts.....	16
6. RESTful API concepts.....	17
7. Planning	18
8. Development	19
8.1. Architecture	19
8.1.1. Code Architecture	19
8.1.2. App Architecture.....	20
8.2. Design	21
8.3. Implementation	26
8.3.1. Use case diagram	26
8.3.2. Technologies used.....	27
8.3.2.1. SignalR.....	27
8.3.2.2. ButterKnife.....	28
8.3.2.3. Retrofit2	28
8.3.2.4. AndroidPdfViewer.....	28
8.3.2.5. Other Technologies.....	28
8.3.3. Secure connection	29
8.3.4. Login.....	30
8.3.4.1 Textual Use Cases	30
8.3.4.2. REST requests.....	32
8.3.5. Agenda	35
8.3.5.1. Textual Use Cases	35

8.3.5.2. REST Requests	37
8.3.6. Voting	40
8.3.6.1. Textual Use Cases	40
8.3.6.2. REST Requests	40
8.3.7. Voting Summary	44
8.3.7.1 Textual Use Cases	44
8.3.7.2. REST Requests	46
8.3.8. Video Display	48
8.3.8.1. Textual Use Cases	48
8.3.8.2. REST Requests	48
8.3.9. Request to speak	50
8.3.9.1 Textual Use Cases	50
8.3.9.2. REST Requests	51
9. Conclusions and future work	53
10. References	54

List of figures

Figure 1. Chamber layout.....	6
Figure 2. Bosch Dicontis device.....	7
Figure 3. Splash Activity XML example	10
Figure 4. [2] Activity lifecycle	12
Figure 5. [3] Fragment lifecycle	13
Figure 6. [4] Service lifecycle.....	14
Figure 7. Shared Preferences variables initialization	15
Figure 8. Shared Preferences creation.....	15
Figure 9. Putting Information in the Shared Preferences	15
Figure 10. SharedPreferences storing file.....	15
Figure 11. Asymmetric PKI system.....	16
Figure 12. JSON example	17
Figure 13. MVC architecture	19
Figure 14. Architecture of the application	20
Figure 15. Splash Screen	21
Figure 16. Main Activity	21
Figure 17. Main Activity with request to speak placed.....	22
Figure 18. Options menu	22
Figure 19. Voting Summary.....	23
Figure 20. Document reader.....	23
Figure 21. Video Display.....	24
Figure 22. Voting Activity.....	24
Figure 23. Reconnect while voting motion on course	25
Figure 24. Use case diagram of a council member	26
Figure 25. SignalR communication server-client	27

Glossary

SSM Scytl System Management

APK Android Application Package

SSL Secure Sockets Layer

HTTPS Hypertext Transfer Protocol Secure

OS Operating System

XML Extensible Markup Language

TLS Transport Layer Security

PKI Public Key Infrastructure

REST Representational state transfer

API Application Programming Interface

JSON JavaScript Object Notation

SOAP Simple Object Access Protocol

UI User Interface

CA Certificate Authority

PKCS Public Key Cryptography Standards

APK Android Application Package

1. Introduction

Scytl is an enterprise that has been implementing solutions to the online voting sector since 2001. Therefore, one of their multiple successful projects is SSM (Scytl System Management) which consists of a web application called eParliament.

The National Council of Provinces of South Africa and the Metropolitan Government of Nashville & Davidson County are using the web application SSM on its sessions.

eParliament is a solution for handling in-chamber functions for the parliament of Ghana, such as debate (i.e. microphone) management, voting and video display. This project consists of three parts: hardware/networking, legislation & agenda management and SSM. The hardware/networking part is managed by a local partner, called Josanti. The legislation & agenda management part is handled by another partner, called NextSense, and it is based on a pre-existing webapp software.

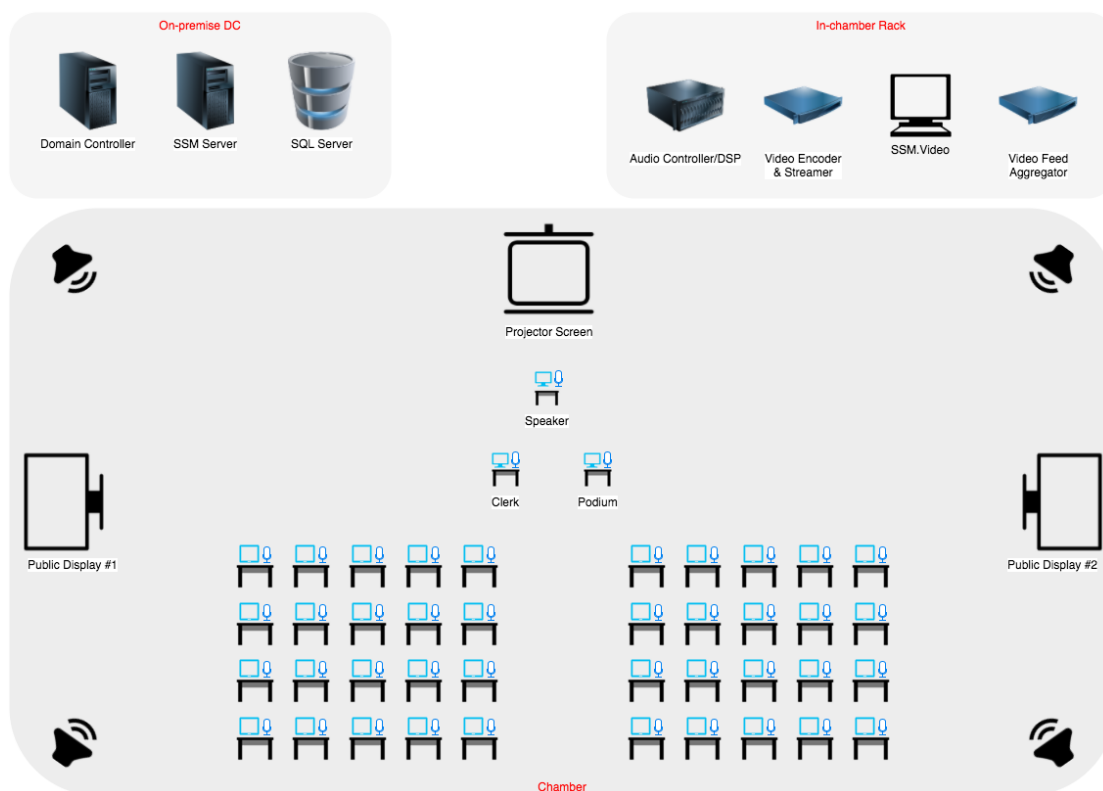


Figure 1. Chamber layout

In 2016, Scytl won a project for the parliament of Ghana. However, due to the infrastructure of the parliament, Scytl decided to migrate the existing web app to Android.

By that time, the Parliament of Ghana had already acquired and installed an audio conferencing system, made by Bosch, called Bosch Dicentis, on each desk of the chamber. These devices have a microphone, a couple of physical buttons and most importantly, an Android tablet embedded into them, running a customized version of Android.



Figure 2. Bosch Dicentis device

ScytI tested the existing web app, but since this was not a plain audio system, but rather a conferencing system, there was a lot of overlap between SSM and Bosch's own software. Furthermore, when the web app was run in the browser of the device, there were two main issues:

1. **Performance:** the performance of the web app on the Bosch tablets was not good enough for such an application.
2. **Manageability:** the SSM web based front-end application was developed to use client-side SSL certificates to allow the server to identify the device from which a request is coming from; this is a critical function of SSM, as it allows it to properly update the graphical chamber view and dynamically display who is sitting where. Installing and updating SSL certificates on 300 Android tablets would be a big problem. Furthermore, inherent Android restrictions force the user to have to define some sort of passcode for accessing the tablet if client credentials are to be used, which means that this would need to be configured for each unit separately.

For these and other reasons, it was decided that the web based version of the SSM front-end is not applicable for this chamber and they decided to develop a native Android client for SSM. Bosch offers a facility that allows the system administrator to deploy an APK (Android Application Package) to all the tablets with one mouse click, so this makes sense from an administrative perspective. Furthermore, using a native application provided more flexibility to allow the server to identify each device uniquely.

2. Objectives

The final objective of the project is to develop an Android application able to manage a real-time parliamentary session, having as a very important point the security component.

The parliament of Ghana is composed by the president, vice president, speaker of Parliament, clerk of the parliament and the council members of the parliament. However, the final users of the application are the council members.

The main objectives to accomplish this goal are:

- **Secure Connection**

The app must be secure, all the connections with the server must be HTTPS, signed with a certificate for the server and another one for every different device that is using the app.

- **Login**

The user must be able to login with their previously assigned credentials.

- **Agenda**

The user must be able to check the agenda which specifies the points that will be discussed during the session. Also, the user must be able to see in detail each point of discussion. If it has a document attached, the user must be able to open it inside the app.

- **Voting**

When the point of discussion requires to vote, the user must be prompted with a voting screen and be able to submit their vote.

- **Voting Summary**

The user must be able to check the summary of all times votes of the parliament, being able to filter it by date. The results can be paginated.

- **Video Display**

If there is an ongoing streaming video, the user must be able to watch it.

- **Request to Speak**

The user must be able to request a speaking turn during the assembly.

3. Basic Parliamentary Concepts

This project will employ the following parliamentary concepts:

- **Session:** It is a meeting or a group of meetings which expose one or more points of discussions.
- **Agenda:** Specify the points of discussions that will be debated during a session.
- **Motion:** It is a proposal made by a member of the chamber, to try to decide about a point of discussion.
- **Submit vote:** The Members of the chamber submit their votes to the motions proposed.
- **Council:** Group formed by the representatives of the parliamentary chamber. They have the mission to discuss the points of interest and take decisions on them.
- **President:** President of the parliamentary chamber. Leads and conducts the session.

4. Basic Android concepts

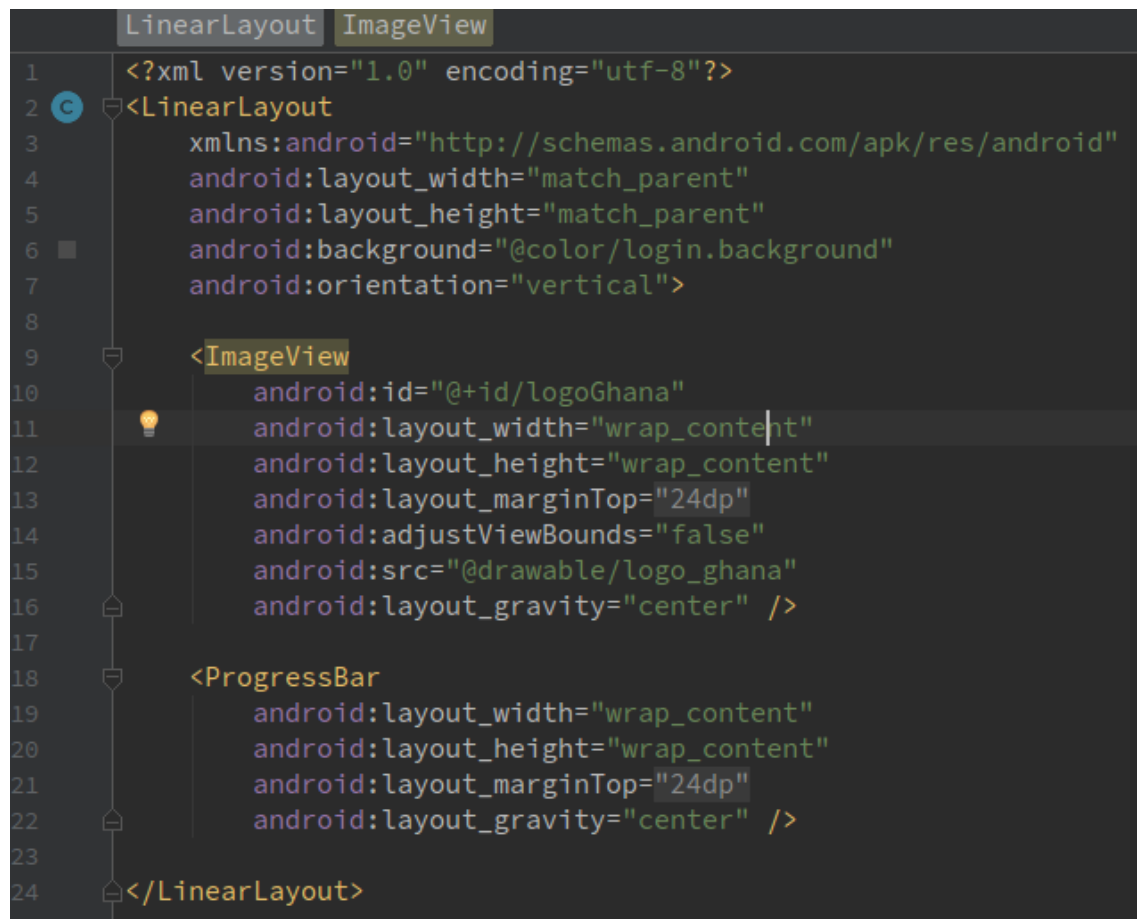
Android is a Linux-based OS developed by Google, it works in a wide variety of devices, but it is specially optimized to work on devices with touchscreens as smartphones or tablets.

Before continuing, it's important to understand some basics concepts of Android developing. This section will give a brief explanation of the ones used in this project.

4.1. Activity

An activity [1] is an Android application component that represents a single screen of the app with the interface that the user will interact with.

Each activity has its own layout, specified in an XML file, which represents the view the user will interact with and its own controller, a Java class, which dictates how the view reacts to the user interaction.



```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      android:background="@color/login.background"
7      android:orientation="vertical">
8
9      <ImageView
10         android:id="@+id/logoGhana"
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:layout_marginTop="24dp"
14         android:adjustViewBounds="false"
15         android:src="@drawable/logo_ghana"
16         android:layout_gravity="center" />
17
18     <ProgressBar
19         android:layout_width="wrap_content"
20         android:layout_height="wrap_content"
21         android:layout_marginTop="24dp"
22         android:layout_gravity="center" />
23
24  </LinearLayout>
```

Figure 3. Splash Activity XML example

An app, usually, is made of multiple activities, working together to form a cohesive user experience, although every activity is independent of each other. As well, when a new activity is started, the previous one is saved in a stack (Back Stack) but not being destroyed, just stopped and then when

the Back Button is pressed, the current activity is destroyed and the last activity of the stack comes forward.

The first activity presented to the user when the app is opened is called the "Main Activity", however, eParliament has the login Activity where the user does the login, that this can be called as well as the "Main Activity" and the "Main Activity" by itself that is launched when the user has successfully logged in.

Furthermore, there is a "special" kind of activity called "Splash Activity". This is a temporary activity shown meanwhile the app prepares to launch. This screen, usually simply shows the logo of the app and do some pre-required tasks at the background to the proper work of the app later.

4.1.1. Life Cycle

The activity lifecycle [2] is the set of states an activity has during its entire lifetime, from creation to destruction. As the user interacts with the app, the different activities move into different states.

An activity can be in three main states:

- Resumed. The activity is in foreground and the user can interact with it.
- Paused. Another activity is foreground but this one is still visible.
- Stopped. The activity is running in background and is no longer visible.

While the activities change among these states, different callbacks are getting called.

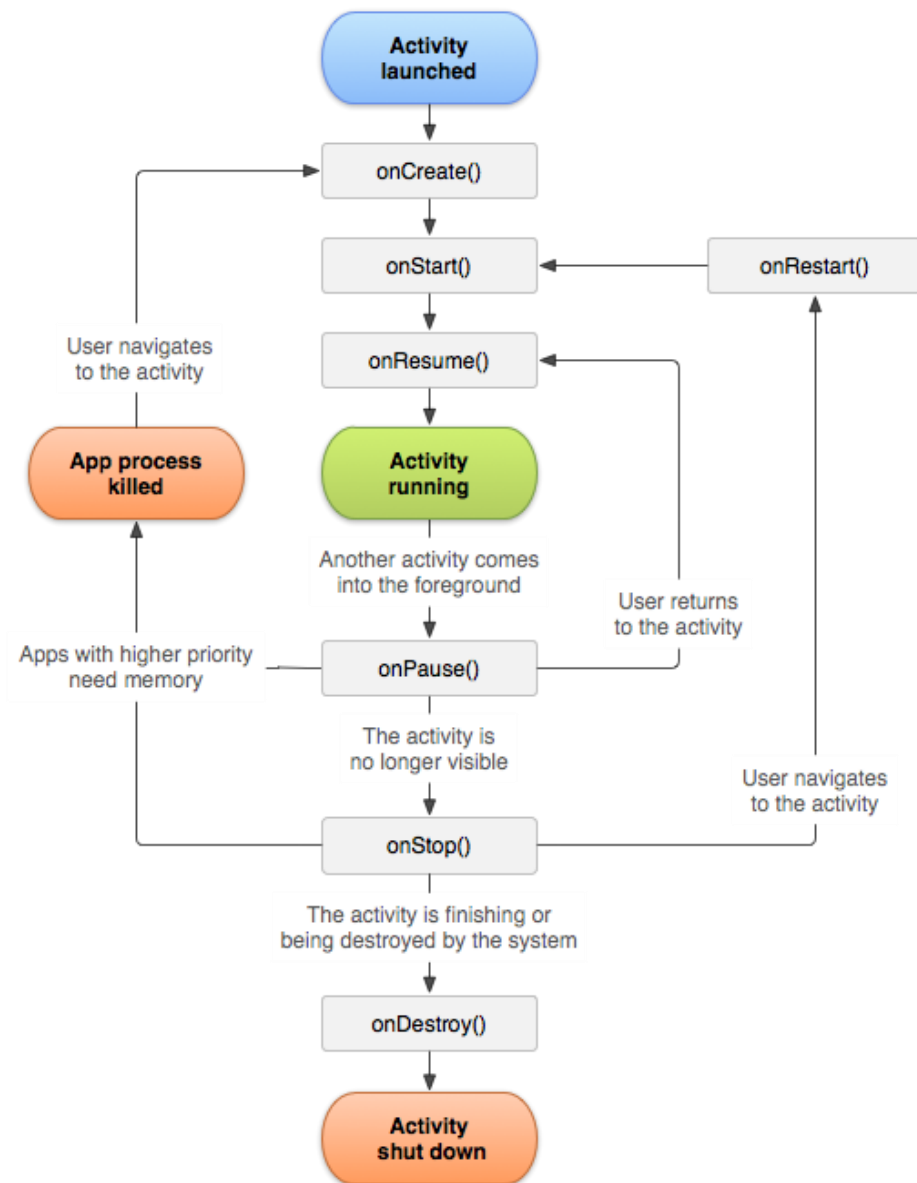


Figure 4. [2] Activity lifecycle

4.2. Fragment

A Fragment [3] is a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running. Multiples fragments can be combined in a single activity.

eParliament interface implementation is fragment-based, since it provides a more dynamic design.

4.2.1. Lifecycle

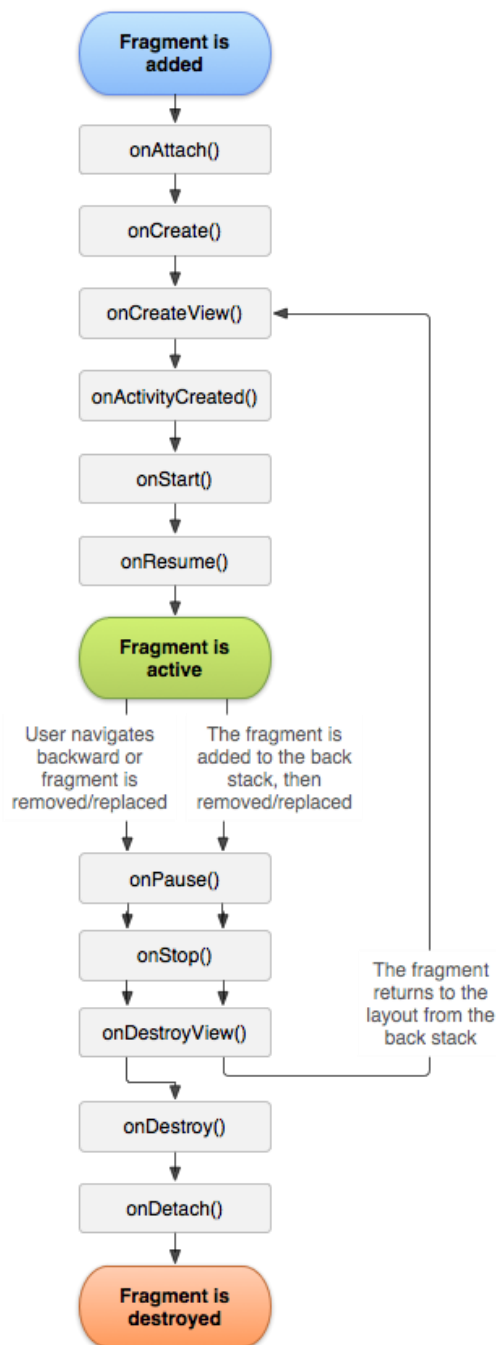


Figure 5. [3] Fragment lifecycle

4.3. Service

A Service [4] is an application component representing an application's desire to perform a long-running operation in the background while not interacting with the user.

There are two ways to start a service:

- **StartService.** This means that the service is called by an activity or fragment to start a service indefinitely in the background.
- **BindService.** eParliament uses this way to start service, because it offers a way to interact the interface with the service.

4.3.1. Lifecycle

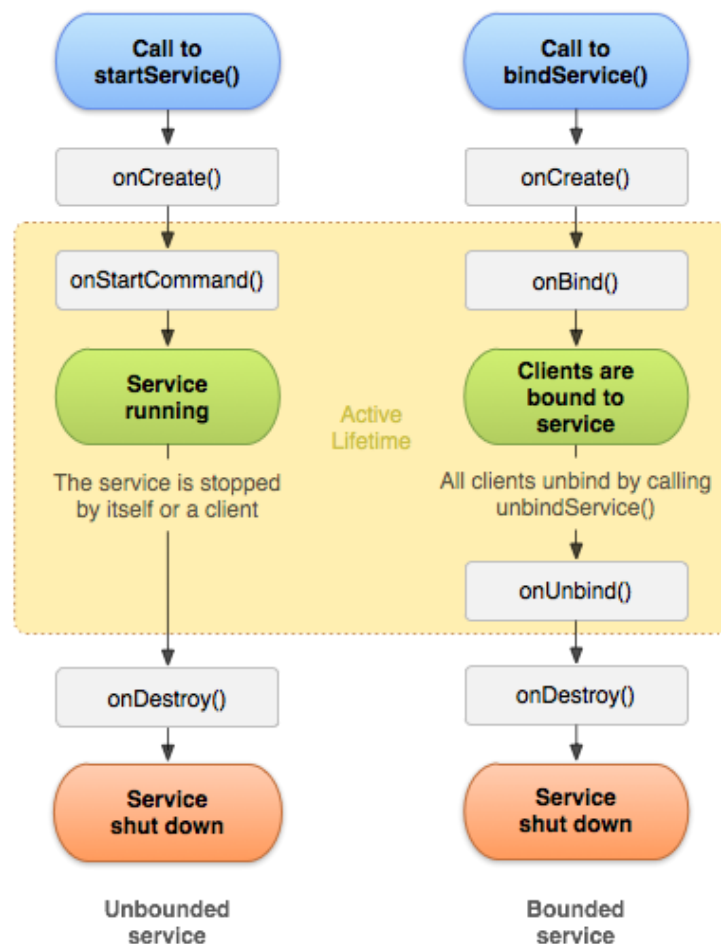


Figure 6. [4] Service lifecycle

4.4. Shared preferences

Android provides many ways of storing data of an application. One of this way is called Shared Preferences [5]. Shared Preferences allow you to save and retrieve data in the form of key,value pair.

The SharedPreferences class provides a general framework that allows us to save and retrieve persistent key-value pairs of primitive data types. We can use SharedPreferences to save any primitive data: Booleans, floats, int, long and strings. This data will persist across user sessions, even if the application is closed or killed.

For example, to save the token of the logged user:

```
1 //Define the shared preferences name
2 public static final String Prefs_NAME = "API_PREFS"
3 //Define the string of the variable that will save
4 public static final String Prefs_TOKEN = "TOKEN"
```

Figure 7. Shared Preferences variables initialization

```
1 //Create shared preferences for the user
2 private SharedPreferences sessionPrefs = new SharedPreferences(context);
```

Figure 8. Shared Preferences creation

```
1 //Save the token in the shared preferences
2 SharedPreferences.Editor editor = context.getSharedPreferences(Prefs_NAME, Context.MODE_PRIVATE);
3 editor.putString(Prefs_TOKEN, "token-encrypted-key");
4 editor.apply();
```

Figure 9. Putting Information in the Shared Preferences

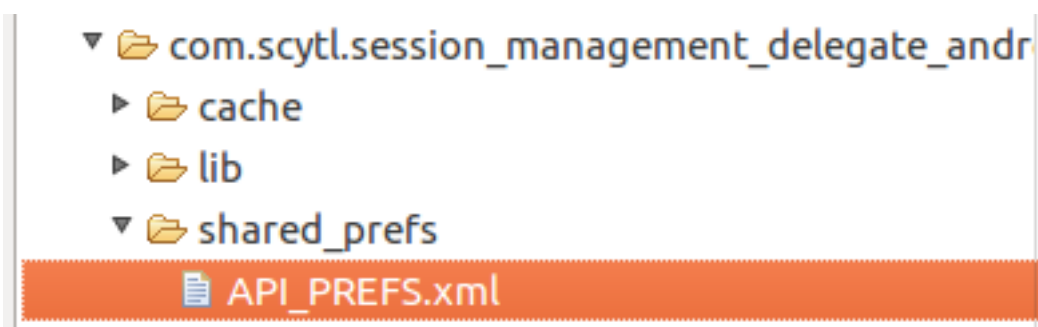


Figure 10. SharedPreferences storing file

5. Basic Security concepts

One of the important concerns of eParliament is being secure, since the app has to ensure that the decisions made follow the democracy standards, like its analogic predecessor did.

For this purpose, it's important to understand how the HTTPS certificates work.

HTTPS [6] uses one or two secure protocols to encrypt communications - SSL (Secure Sockets Layer) or TLS (Transport Layer Security). Both of them use an 'asymmetric' Public Key Infrastructure (PKI) system.

An asymmetric system uses two keys to encrypt communications, a public key and a private key. All that a public key encrypts can only be decrypted by the private one, and all that is encrypted with the private key has to be decrypted with the the public one.

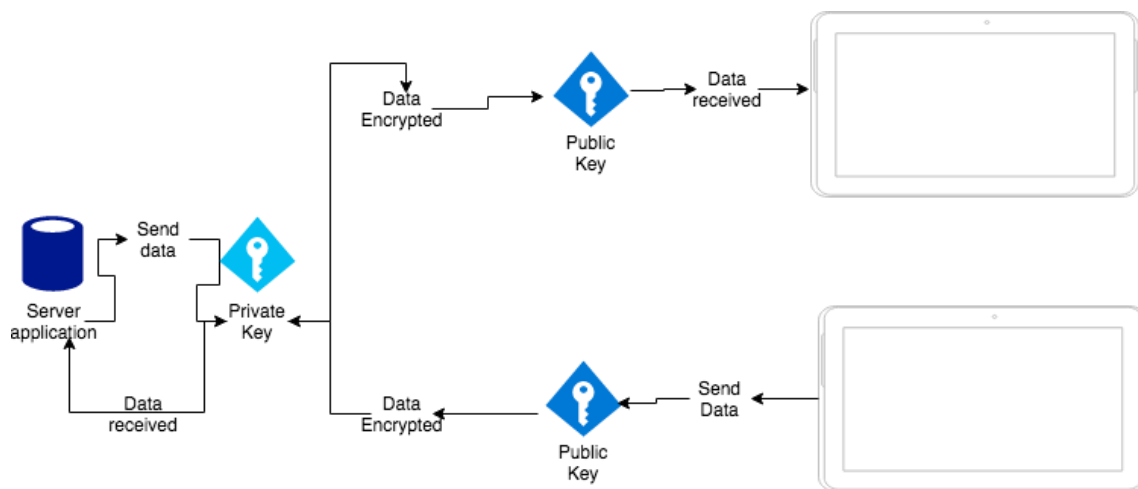


Figure 11. Asymmetric PKI system.

The private key should only be accessible to their owner. Instead, the public key is intended to be distributed to anybody that needs to be able to decrypt information that was encrypted with the private key.

For this project, it is used SSL certificates, which can have the following extensions:

- **PKCS#12 (.p12)** It is the only extension able to import and export certificates and their private key in the same file.
- **.CRT or .CER.** These extensions, distributed by a certificate authority (CA), are used to verify authenticity.
- **.PEM.** It can be read with any text editor. It is the most dispensed format by CAs
- **.DER.** This format is a variation of the .PEM, encoding the file in a binary format instead the ASCII format of the .PEM

eParliament uses a .crt certificate to validate the connection with the server, and a .p12 certificate in order to validate each device, which is included with each call made to the SSM back-end, so that the server knows which specific device (and therefore which desk/seat in the room) each request comes from.

6. RESTful API concepts

Once we have successfully made the connection with the server. It is the time to make requests to it. To achieve this, eParliament uses a RESTful API. A RESTful API (Application Program Interface) is able to perform HTTP requests as:

- **GET.** Retrieve information from a server
- **PUT.** Update data from the server
- **POST.** Deliver data to the server.
- **DELETE.** Remove data from the server.

The data transfer between the app and the server is done with JSON objects.

JSON [7] (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. JSON is a text format that is completely language independent, but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. Therefore, a JSON is easy to understand, comprehensive and easy to use through all the overall languages.

```
{
  "id":12313,
  "name":"Chamber1",
  "members":[
    {
      "name":"ChamberMember1",
      "id":111
    },
    {
      "name":"ChamberMember2",
      "id":112
    }
  ]
}
```

Figure 12. JSON example

In order to use REST requests in android, one of the best practices is to make the calls asynchronously. This means that the call will be made in another thread and not in the main UI thread. This is recommended because if it is done in the main thread, it would lock the screen for some seconds and the user would think that the application is not responding. To use asynchronous or synchronous tasks depends on the task by itself and for what you want to use it.

For example, if it is a login request, it can't enter into the application if it is not logged. So, meanwhile it is being logged in, the screen has to be locked or to show a progress bar indicating that the request is being held. In the first case, it can be used a synchronous task, however this is not recommended, therefore, a smooth progress bar is shown with the asynchronous task by behind.

eParliament manages all the REST calls by using asynchronous tasks in order to not lock the screen for the operations or if that task needs to be done first, such as the login, the application uses a progress bar indicating that some operations are being done until it is finished.

7. Planning

Following Scytl standards, the planning of the project was scheduled in different sprints.

A sprint, is the planning of tasks to do during a constant set of time.

The sprints were defined in JIRA, software that helps to schedule tasks to do. It has a visual interface that makes really easy to see the state of the designed tasks.

Each sprint had 15 days of duration. At the start of every sprint were planned the tasks to do during its duration. Each sprint tried to encompass a different module of the project.

This project was finished in the 8th sprint:

1. The first sprint was dedicated to search use cases and to plan the project accordingly.
2. The second sprint was focused on the UI, designing it and developing its layout.
3. The third sprint had some remaining tasks of the second, but it was focused on the security aspect, implementing the server-client SSL connection.
4. The fourth sprint was focused on the data model, all data classes were made and also the REST petitions were implemented.
5. The fifth sprint was about implementing the controllers of the activities.
6. The sixth sprint continued to develop the controllers of the activities.
7. In the seventh sprint was implemented the connection among the controllers, the REST petitions and the data model.
8. The eighth sprint had the objective to realize all the testing the app needed and to fix the bugs found.

At the end of every sprint, there was a meeting with the project manager and the web app developer to have perspective how the project is going and to plan the next sprint.

8. Development

8.1. Architecture

8.1.1. Code Architecture

eParlament follows the MVC pattern.

The MVC is a pattern that separates your application into 3 sets of responsibilities:

- **Model**

The model is the data structure of the application, specifying his business logic. It is independent of the view or the controller, what makes the code more reusable.

- **View**

The view is the representation of the data defined in the Model. The view has the responsibility to show the user interface and capture their interactions and communicate them to the controller. The view has no idea what the application has to do when the user interacts with it, it is out of his responsibility.

- **Controller**

The controller communicates the Model and the View. Is the one with the responsibility to react to the interactions of the user that the View captures. It is also responsible to update the view when the data requires it.

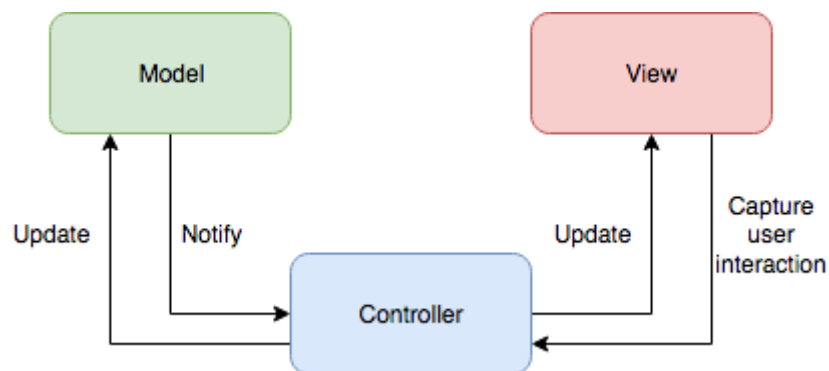


Figure 13. MVC architecture

There are other patterns in order to structure the project's code, such as MVP (Model-View-Presenter), MVVM (Model-View-ViewModel).

MVC was chosen because even that it separates the code of the app in 3 different independent modules, which causes that a part of the code is reusable, it does not require too many modules or classes as the other patterns do. Probably, if the project was so much bigger, another pattern would be picked.

MVC has a problem with big projects, since, if a view has lots of functionalities, it can generate "massive" view controllers and make it harder to maintain. But, since it's not the case, MVC was picked, due to its simplicity.

8.1.2. App Architecture

This is the final architecture, adding the REST server calls using the library Retrofit and the validation with the server through SSL certificates.

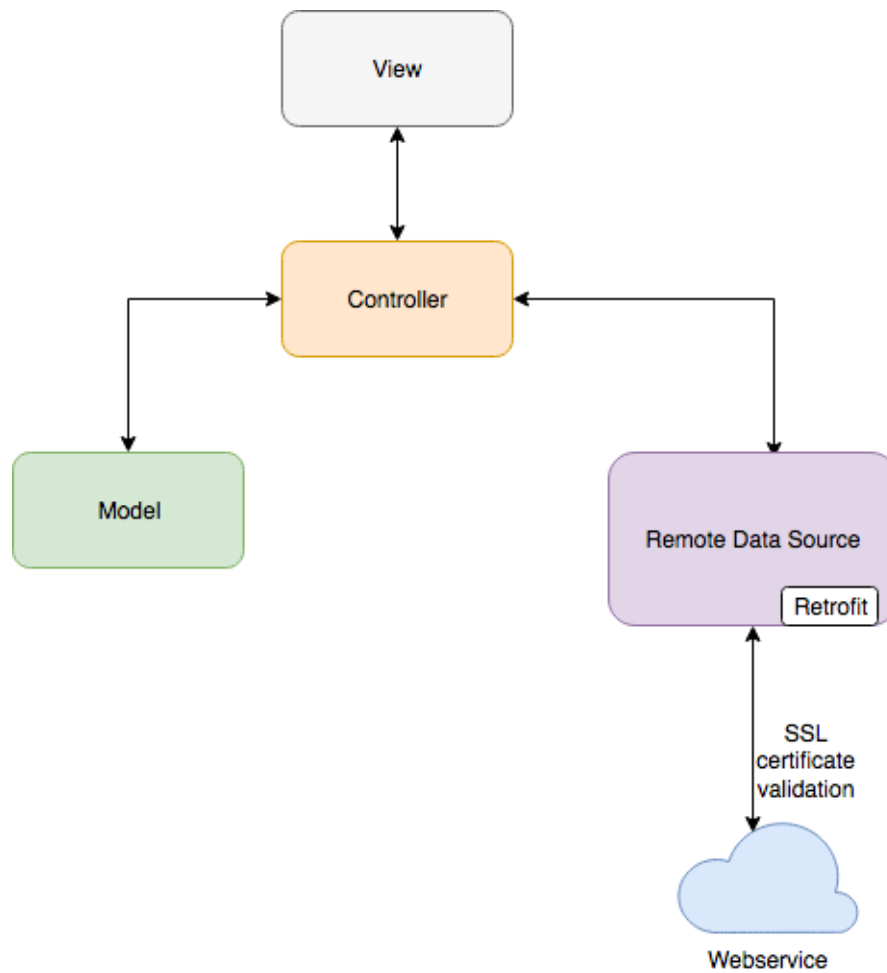


Figure 14. Architecture of the application

8.2. Design

The application design is divided in four important components:

1. Splash Activity:

It is the first screen that the user sees, meanwhile, in the background, the system does all the necessary operations needed before entering the app. In this case, the system makes the server authentication and the device certificate validation in order that after, the needed requests to the server can be done.

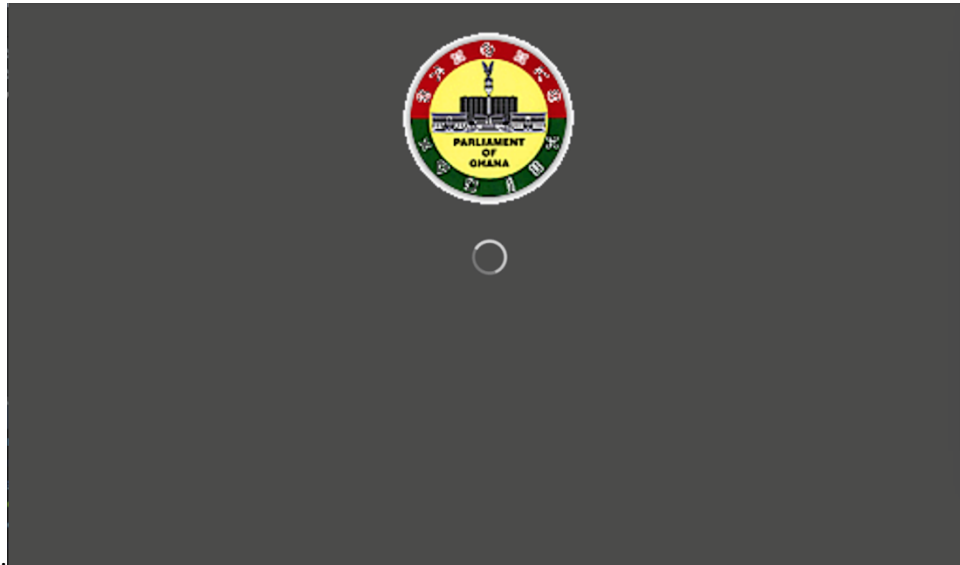


Figure 15. Splash Screen

2. Main Activity:

This is the main application interface which in the app, it is designed with a navigation drawer and its content is displayed with a fragment. Every time the user enters to the app and has successfully logged in, they will see this screen.

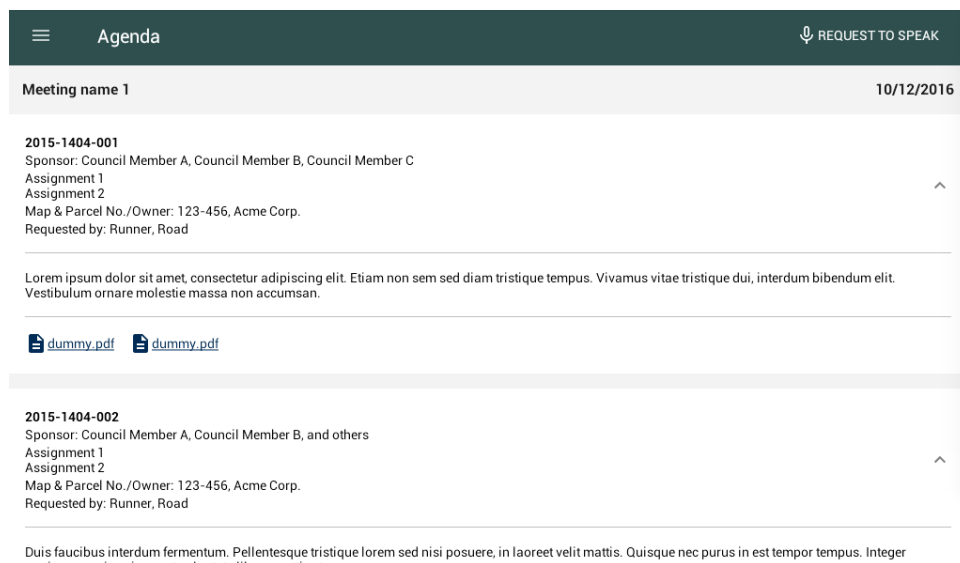


Figure 16. Main Activity

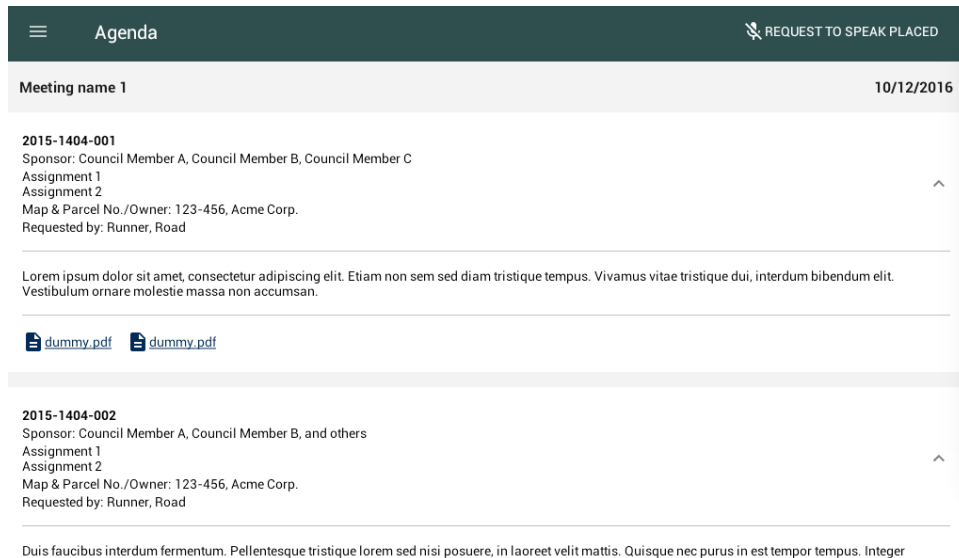


Figure 17. Main Activity with request to speak placed

3. Options Menu:

The app has a left lateral menu formed of a navigation view component, which contains all the different options available in the menu. When one of them is pressed, the fragment contained in the main activity changes to the corresponding option selected. To use a fragment gives versatility to the app when there is an update of the data shown.

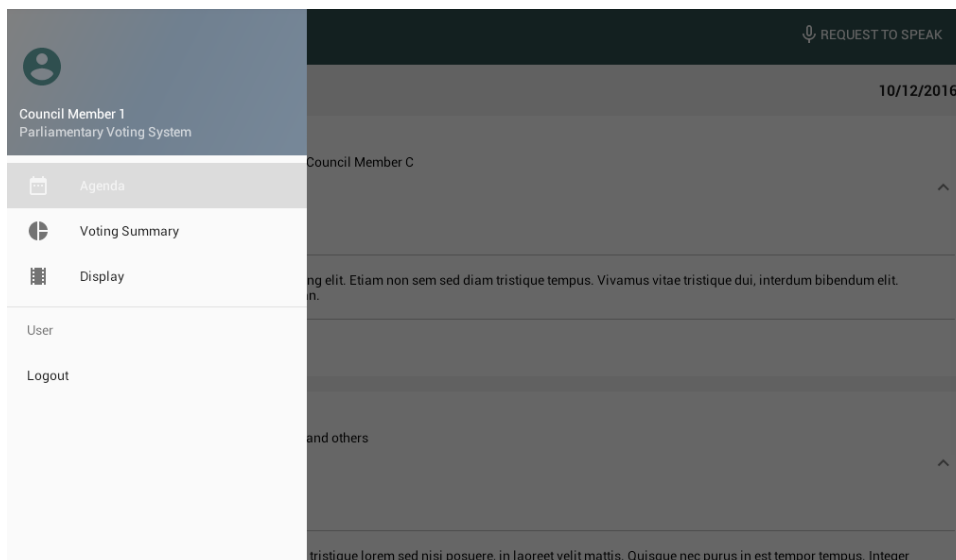


Figure 18. Options menu

4. List Results:

The app contains 2 main lists:

- Agenda View (list of agendas)
- Voting summary view (list of results)

Android offers you two ways to implement this, with a RecyclerView or a ListView. However, RecyclerView is more efficient by default, its animations are simpler, layouting is easier to use and the API is much nicer. Therefore, RecyclerView is used for displaying those lists and each item of the list is displayed by a CardView to be able to expand or collapse it.

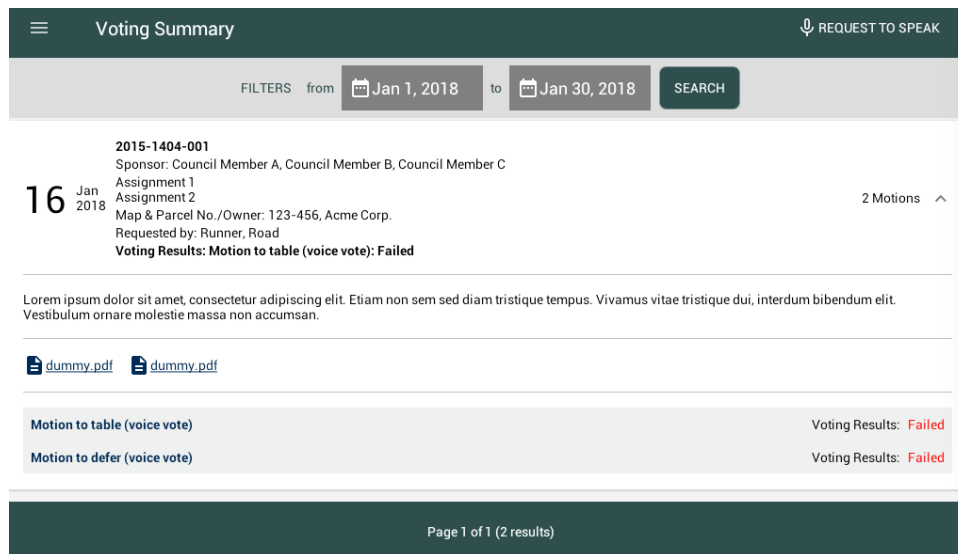


Figure 19. Voting Summary

Apart from this main component, eParliament also has a view that allows the user to read a document, and another one that shows a streaming video.

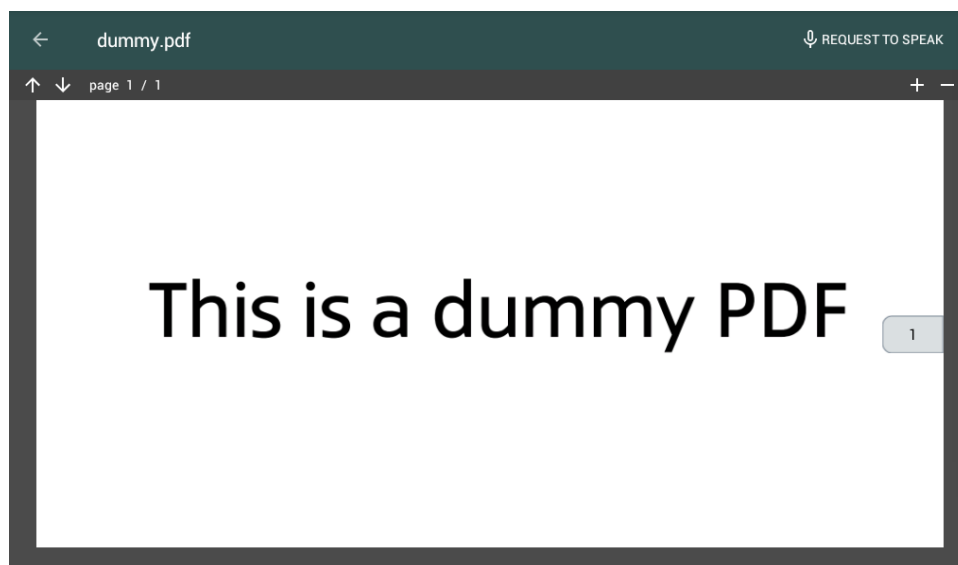


Figure 20. Document reader.

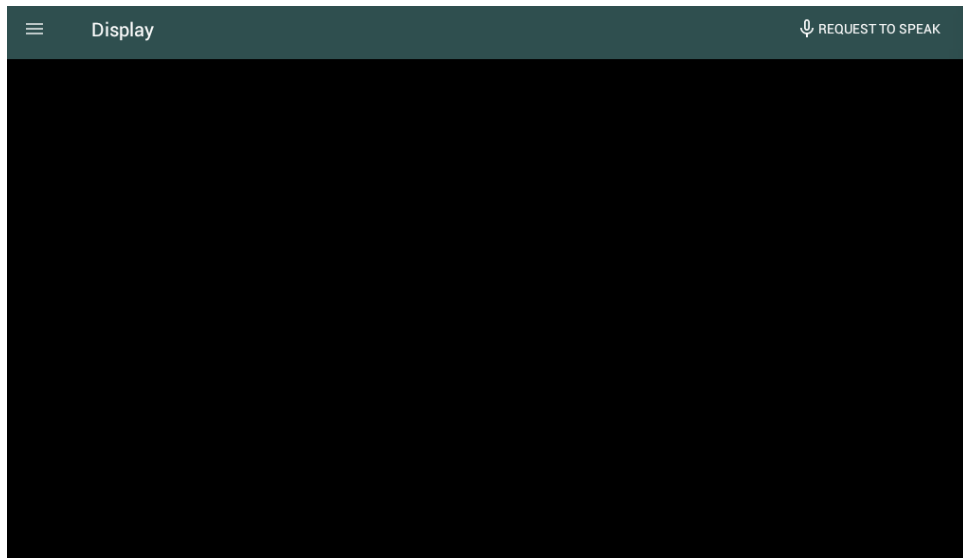


Figure 21. Video Display

Finally, eParliament has a view that is only accessible when there is a voting motion in progress where the user can submit their vote.

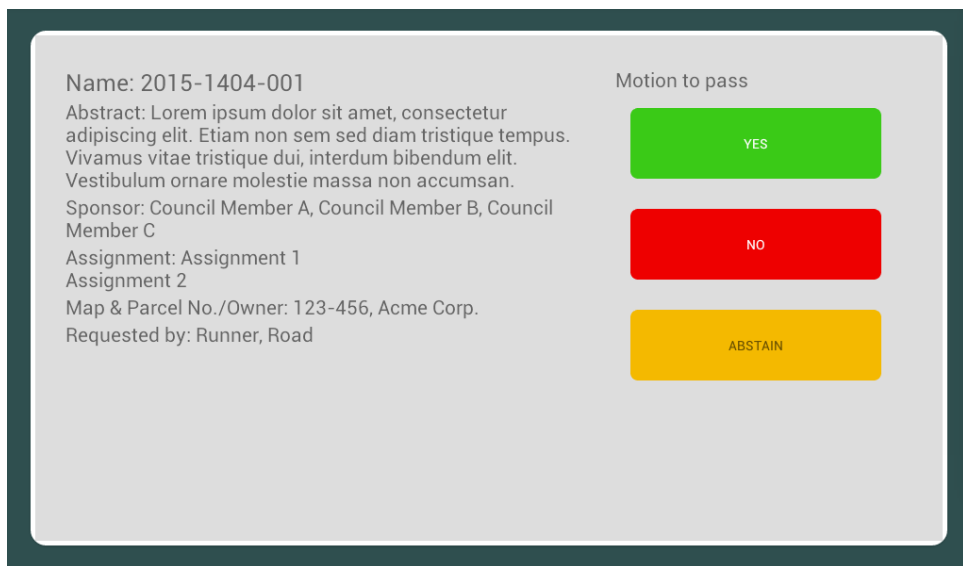


Figure 22. Voting Activity

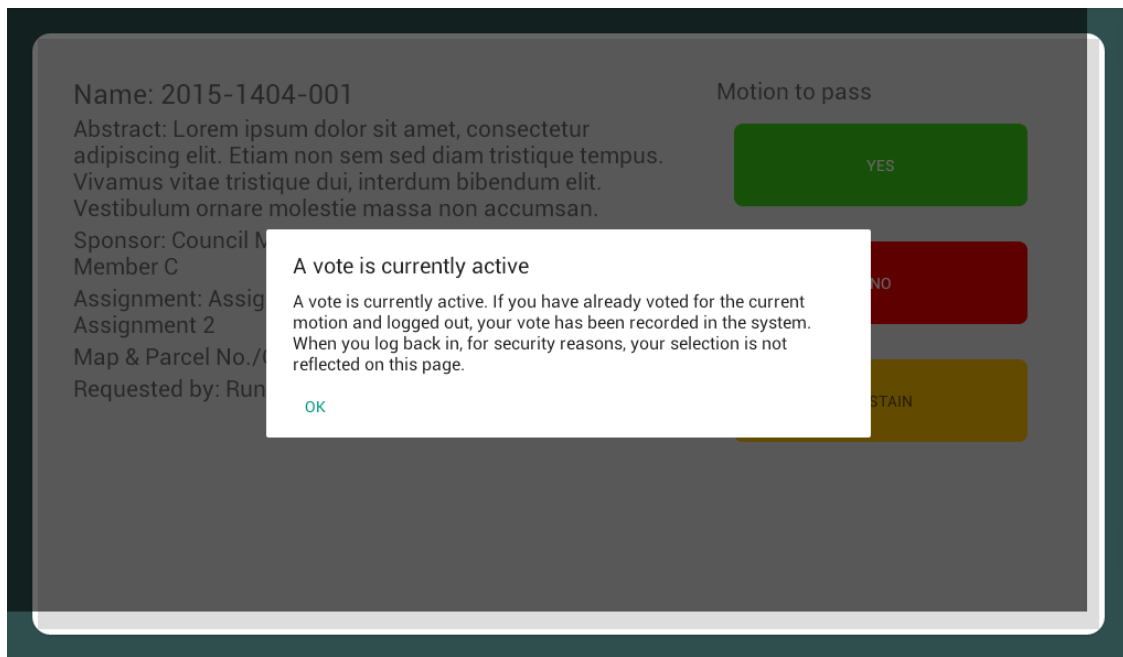


Figure 23. Reconnect while voting motion on course

8.3. Implementation

To accomplish all the objectives demanded in this project, the first step was to find all the use cases that it was needed to fulfil. This is the use case diagram that specify them:

8.3.1. Use case diagram

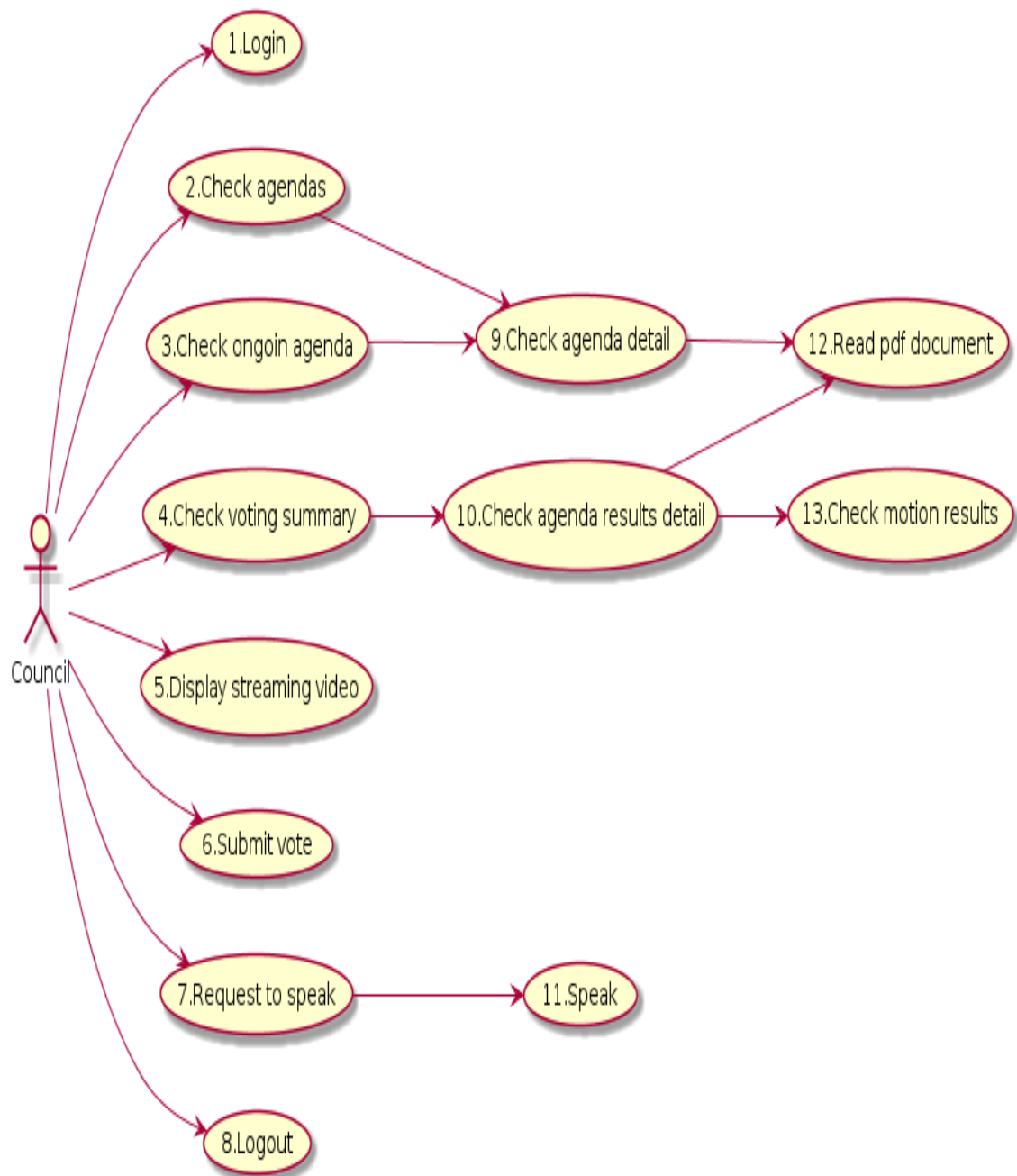


Figure 24. Use case diagram of a council member

8.3.2. Technologies used

The next step was to define the technologies to be used. Since this app is a migration of a web app, it was required to use this library:

8.3.2.1. SignalR

SignalR [8] is a library that simplifies the process of adding real-time communication server-client. It makes possible to push content from the server to its clients instantly as it becomes available, so the client doesn't have to request the new data every undefined amount of time.

This library is used in order to add a real-time connection to the application, for example, when it is the time to vote, the server sends a signal to the application, the application receives it and then the voting activity is shown. The user will be abstracted of pressing any button in order to see that screen, and also the voted answer will be gathered instantly to be able to see the results of the motion.

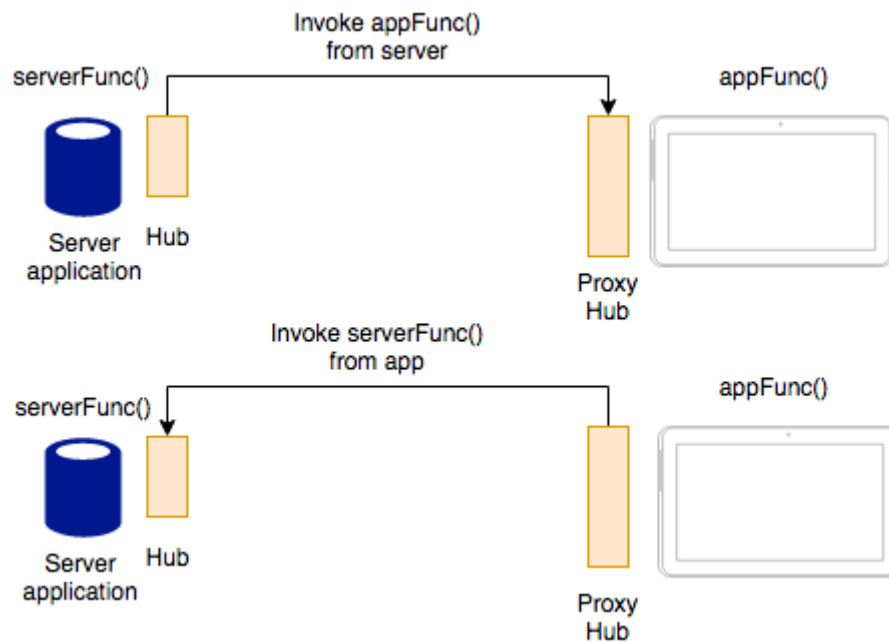


Figure 25. SignalR communication server-client

SignalR has the definition of transports (the sent data) and there are some types: Websockets, Server Sent Events, Forever Frame, Long polling. SignalR by itself decides which technique the server and client access is both support to use.

As the back-end and the web app were configured using this technology, that was the right option to eParlament.

Some other libraries that are used are:

8.3.2.2. ButterKnife

It is a view binding library that abstracts you through annotations the way some basic calls or instantiations of components are made in the Activity, Fragment, Service... This helps to make your code clearer in an easy way to program it and it is a time saving.

8.3.2.3. Retrofit2

Retrofit2 [9] is a library that makes it easy to connect to a REST web service, it allows you to turn the REST API into a JAVA interface, which makes it simple to manage the content retrieved by each request and to make the request itself. Retrofit2 was built on another library called OkHttp, this last library is the one in charge to make the connection with the url and furthermore. Retrofit2 doesn't count with its own JSON parser (from JSON to a Java object). Therefore, you can choose any supported JSON library to use, or to customize your own converter to use it. As well Retrofit2 gives you a smooth API in order to manage asynchronous and synchronous tasks, which it is required for this project. eParliament uses the retrofit:converter-gson to parse JSON to a Java object.

Retrofit2 was chosen due to its ability to parse JSONs in a very simple, elegant, and easy-to read way.

8.3.2.4. AndroidPdfViewer

AndroidPdfViewer as its name by itself says, it helps you to render pdf documents through the same application and it is not needed to go outside the application to see the pdf. As for this project, it was required to see the pdf document in the application, this library was decided to use. There were other native options that android gives you, but it was required to go outside of the application or how the connection made with the server was secured by certificates, it was not easy to see it inside through a WebView. There were other libraries as well, but this one was the most complete in order to customize the rendering of a pdf.

8.3.2.5. Other Technologies

Apart from the libraries mention above and the natives from Android. I also used a program called Postman in order to test the REST API calls.

Regardless that not long ago was launched a new programming language for Android devices called Kotlin, I stuck with Java because, nowadays, it has a larger community which translates into more documentation and more usable libraries that I can have access.

8.3.3. Secure connection

The first step was to define a way to maintain a secure connection of the app with the server. For so, every time the user enters into the application, it goes first to the SplashActivity and in the background it does the connection with the server validating the public certificate (.crt) and the private key for the device (.p12) and then the application is launched. For now, the certificates are inside the program and when it is installed the APK, it already has the certificates.

To make the REST requests secure, once the user logs in, the server retrieves a token for the user. In every call, this token must be passed in the header.

After this, the application has two services configured:

- LoggedService

This service is in charge to validate every time the user enters to the application if it is logged or not. If it is not logged, it shows the login Activity and if it is logged, it enters directly to the application. Apart from this, every minute, it validates if the user is logged or not. When it is said the user is logged, it means that it sends a GET request to validate if the token is still valid or not. As well, every minute validation helps to have the user session actives meanwhile they are logged, because if not, after 15 minutes of no activity, the server by itself would cut the connection and the token would not be valid anymore and this is not wanted.

- SignalRService

This service uses the SignalR library, to first configure a Hub connection with valid credentials (in this case its credentials is the token of the user that it is sent on the connection), then it is configured the HubProxy and the connection is started. Finally, the HubProxy listeners are started in the service to be able to detect any changes/events that there is on the server and when there is a change/event, the listeners receive it and then the UI is changed according to it.

8.3.4. Login

After having a secure channel between the server and the app, it is needed to do the login. These are the use cases on how the user interacts.

8.3.4.1 Textual Use Cases

- This login is made when it is the first time that the user enters into the application.

UC1. Login 1	
Description	Login inside the app.
Actor	Council member
Preconditions: <ol style="list-style-type: none">1. The public certificate of the server is installed.2. The private certificate of the device is installed.3. Private credentials are given to the council member.4. The device has internet connection.5. The user wasn't logged before.	
Steps: <ol style="list-style-type: none">1. The user enters the application.2. The app shows the splash activity.3. The app shows the login screen.4. The user writes its user and password and they press the login button.5. The app authenticates through the server.6. The app stores the obtained token from the server response.7. The app shows the home screen and the user has successfully login.	
Alternative flow: <ol style="list-style-type: none">2.1. Invalid certificate authentication.<ol style="list-style-type: none">2.1.1. The system displays an error message and it goes to point 3.4.1. Wrong credentials, inexistent internet connection.<ol style="list-style-type: none">4.1.1. The system displays an error message and it returns to point 3.4.2. Inexistent internet connection.<ol style="list-style-type: none">4.2.1. The system displays an error message and it returns to point 3.	
Postcondition: <ol style="list-style-type: none">1. The user has successfully logged in.	

- This login is made when the user went out of the application and they have already logged in before and it is no needed to write again the credentials.

UC1. Login 2	
Description	Login inside the app.
Actor	Council member
Preconditions: <ol style="list-style-type: none"> 1. The public certificate of the server is installed. 2. The private certificate of the device is installed. 3. The device has internet connection. 4. The user was logged before and the device has a valid token. 	
Steps: <ol style="list-style-type: none"> 1. The user enters the application. 2. The app shows the splash activity. 3. The app checks if the token is valid. 4. The app shows the home screen and the user has successfully login. 	
Alternative flow: <ol style="list-style-type: none"> 2.1. Invalid certificate authentication. <ol style="list-style-type: none"> 2.1.1. The system displays an error message and it goes to point 3 of UC1 Login 1. 3.1. Invalid Token. <ol style="list-style-type: none"> 3.1.1. The system goes to point 3 of UC1 Login 1. 3.2. Inexistent internet connection. <ol style="list-style-type: none"> 3.2.1. The system displays an error message and it returns to point 3 of UC1 Login 1. 	
Postcondition: <ol style="list-style-type: none"> 1. The user has successfully logged in. 	

- This is when the user does logout.

UC8. Logout	
Description	Logout of the app.
Actor	Council member
Preconditions: <ol style="list-style-type: none"> 1. The user is already logged. 2. The device has internet connection. 3. There is no motion being held. 	
Steps: <ol style="list-style-type: none"> 1. The user tabs on the menu button. 2. The user tabs on the logout tab. 3. The app deletes all the user information. 4. The app logouts the user. 5. The app shows the login activity. 	
Postcondition: <ol style="list-style-type: none"> 1. The user has successfully logout. 	

8.3.4.2. REST requests

- (POST) {SERVER_URL}/ login

Performs a login from a username and a password.

Request arguments:

- **String username.** This argument is required
- **String password.** This argument is required

Header:

- The header in this request is empty

Body example:

```
{
  "username":"username",
  "password":"pass"
}
```

Response:

Body arguments:

- **String User.** User name.
- **Bool IsAuthenticated.** The user is logged in the system
- **String Token.** User token
- **Bool ShowOnScreenKeyboard.** Not used in the app.

Example:

```
{
  "User":"username",
  "IsAuthenticated": true,
  "Token":"123ks123kwenq0932k-qwe",
  "ShowOnScreenKeyboard": false
}
```


- **(GET) {SERVER_URL}/logged**

Checks if a token is valid and from a logged user.

Request arguments:

This service has the argument body empty

Header:

- x-access-token. User access token

Response:

Body arguments:

- **String User.** User name.
- **Bool IsAuthenticated.** The user is logged in the system
- **String Token.** User token
- **Bool ShowOnScreenKeyboard.** Not used in the app.

Example:

```
{
  "User": "username",
  "IsAuthenticated": true,
  "Token": "123ks123kwenq0932k-qwe",
  "ShowOnScreenKeyboard": false
}
```

- **(POST){SERVER_URL}/logout**

Logout a user by their access token

Request arguments:

This service has the argument body empty

Header:

- x-access-token. User access token

Response:

The response is empty

- **(GET){SERVER_URL}/roles**

Get the role that a user has, by their access token

Request arguments:

This service has the argument body empty

Header:

- x-access-token. User access token

Response:

Body arguments:

- **String Name.** Role of the user.

Example:

```
{  
  "Name": "Delegate"
```

8.3.5. Agenda

This is the main view that the user sees when they have already logged in successfully. The menu agenda, it is formed by meetings. Each meeting has many points/topics to be discussed during the session. When there is not any active meeting, the application will show all the meetings programmed for the session, but when there is an active meeting. The application will show only that meeting with its points to discuss. As well, when there is an active discussion point, the application will show a bottom bar, indicating which is the active discussion point, and when you click on it, it will redirect you to it.

8.3.5.1. Textual Use Cases

- This is the main view of the agenda, that appears when a user is logged in, however, if the user is in another view, this is how the user can access to it.

UC2. Check agendas	
Description	Check the agendas.
Actor	Council member
Preconditions: <ol style="list-style-type: none">1. The user is already logged.2. The device has internet connection.	
Steps: <ol style="list-style-type: none">1. The user tabs on the menu button.2. The user selects the agenda tab.3. The app obtains the agenda through the server.4. The app shows the agenda activity.5. The app shows the agendas.	
Postcondition: <ol style="list-style-type: none">1. The user sees the agendas.	

- If the user wants to check the ongoing discussion point.

UC3. Check ongoing agenda	
Description	Check the currently agenda/point/subject that it is being discussed.
Actor	Council member
Preconditions: <ol style="list-style-type: none">1. The user is already logged.2. The device has internet connection.3. There is an active agenda that is being discussed.	
Steps: <ol style="list-style-type: none">1. The app displays a bottom bar, showing the currently discussed agenda.2. The user tabs on the bottom bar.3. The app shows the agenda activity.4. The app displays the current discussed agenda in detail.	
Postcondition: <ol style="list-style-type: none">2. The user sees the current discussed agenda and its detail.	

- If the user wants to see in detail one of the points of the meeting.

UC9. Check agenda in detail	
Description	The user wants to see one point of discussion(agenda) in detail.
Actor	Council member
Preconditions: <ol style="list-style-type: none"> 1. The user is already logged. 2. The device has internet connection. 3. The user is already in the agenda tab. 	
Steps: <ol style="list-style-type: none"> 1. The user tabs on the agenda they want to see in detail. 2. The agenda is expanded to show in detail. 	
Postcondition: <ol style="list-style-type: none"> 1. The user sees the agenda in detail. 	

- The user wants to see a pdf document of the details.

UC12. Read pdf document 1	
Description	The user wants to read a pdf document.
Actor	Council member
Preconditions: <ol style="list-style-type: none"> 1. The user is already logged. 2. The device has internet connection. 3. The user is already in the agenda tab. 4. The user has already tabbed an agenda and it displays its detail. 5. Its detail contains a document. 	
Steps: <ol style="list-style-type: none"> 1. The user selects one document of the agenda detail. 2. The app asks for the document to the server. 3. The app opens the document activity. 4. The app displays the requested document. 	
Postcondition: <ol style="list-style-type: none"> 1. The user reads the document. 	

8.3.5.2. REST Requests

- (GET) {SERVER_URL}/agendaItems

Obtains all agendas in the session of the user, by this access token

Request arguments:

This service has the argument body empty.

Header:

- x-access-token. User access token.

Response:

Body arguments:

- **List<AgendaItem> AgendaItems.** List of agendas of the session.
An AgendaItem has the following attributes:
 - **String abstractText.** Explaining text of the agenda item.
 - **String assignments.** Assignments of the agenda item.
 - **List<Document> documents.** List of documents contained in the agenda item.
 - **String id.** Id of the agenda item.
 - **String name.** Name of the agenda item.
 - **String owner.** Name of the owner of the agenda item.
 - **String requestedBy.** Name of the requester of the agenda item
 - **Integer sequence.** Not used in the app
 - **Sponsor.** Sponsor of the agenda item
 - **State.** State of the agenda item
- **String Date.** Date of the agenda.
- **String Name.** Name of the agenda.
- **Integer state.** Current state of the agenda.

Example Response:

```
{
  "AgendaItems": [
    {
      "abstractText": "Lorem ipsum dolor sit amet, consectetur",
      "assignments": "Assignment 1/nAssignment 2",
      "documents": [],
      "id": "6155",
      "name": "2015-1494-001",
      "owner": "123-456, Acme Corp.",
      "requestedBy": "Runner, Road",
      "sequence": 1,
      "Sponsors": "Council Member A, Council Member B",
      "state": 1
    }
  ],
  "Date": "10/12/2016",
  "Name": "Meeting name 1",
  "state": 1
}
```

- (GET) {SERVER_URL}/agendaItems/{agendaId}/documents/{docId}/token

Get the token of a document, specify by docid, contained in an agenda, specify by agendaID.

Request arguments:

This service has the argument body empty.

Header:

- x-access-token. User access token.

Response:

Body arguments:

- **String Token.** Token of the document.

Example:

```
{
  "Token": "29c43882-af35-47c9-8bc7-9043fd29ba70"
}
```

- **(GET) {SERVER_URL}/agendaItems/{agendaId}/documents/{docId}**

Get the document, specify by docid, contained in an agenda, specify by agendaID. It has to send the valid token of the document to verify the download

Request arguments:

This service has the argument body empty.

Header:

- x-access-token. User access token.
- token. Token of the document

Response:

Body arguments:

- **ResponseBody responseBody.** Retrofit's responseBody that permits to download the document

8.3.6. Voting

When it is time to vote. The application will receive the signal through SignalR and the application will open the motion activity. The different options to vote are received from the server, as well its corresponding colours to display it in the view.

8.3.6.1. Textual Use Cases

- When there is a current motion, the user can't go to any other screen and the main screen by default will be the motion activity.

UC6. Submit vote	
Description	Submit a vote of the current motion held.
Actor	Council member
Preconditions: <ol style="list-style-type: none">1. The user is already logged.2. The device has internet connection.3. A voting motion is being held.	
Steps: <ol style="list-style-type: none">1. The app shows the voting motion activity.2. The user selects their decision.3. The app sends the decision to the server.	
Postcondition: <ol style="list-style-type: none">1. The user has successfully vote.	

8.3.6.2. REST Requests

- **(GET) {SERVER_URL}/voting/options**

Get the options that the active voting motion has.

Request arguments:

This service has the argument body empty.

Header:

- x-access-token. User access token.

Response:

Body arguments:

- **Bool isSecretVoting.** Indicates if the voting is secret
- **List<VoteOptions> voteOptions.** List with all the possible voting options.

A VoteOption is formed by:

- **String Color,** Color to display with the option.
- **Double Id.** Id of the option
- **String Name.** Text of the option
- **String TextColor.** Color of the text of the option

Example:

```
}
  "isSecretVoting": false,
  "voteOptions": [
    {
      "Color": "green",
      "id": 12312312312,
      "name": "YES",
      "textColor": "white"
    },
    {
      "Color": "red",
      "id": 12312312313,
      "name": "NO",
      "textColor": "white"
    },
    {
      "Color": "yellow",
      "id": 12312312314,
      "name": "ABSTAIN",
      "textColor": "white"
    }
  ]
}
```

- (POST) {SERVER_URL}/voting/castVote

Send a vote defined by his id to a currently active voting motion.

Request arguments:

- **Double VoteOptionsId.** Id of the vote selected.

Example:

```
{  
    "VoteOptionsId": 12312312313  
}
```

Header:

- x-access-token. User access token.

Response:

The response is empty

- (GET) {SERVER_URL}/system/state

If there is any, returns the information of a current active voting motion.

Request arguments:

This service has the argument body empty.

Header:

- x-access-token. User access token.

Response:

Body arguments:

- **Int activeAgendaItemID.** Id of the active agenda
- **Int activeMeetingId.** Id of the active meeting.
- **Int activeMotionID.** Id of the active motion.
- **Bool IsClosing.** Not used in the app.
- **Bool isTie.** Not used in the app.
- **List<String> Roles.** Roles participating in the motion.
- **Int VideoState.** State of the streaming video of the motion.

Example response:

```
{
  "activeAgendaItemId": 9213812,
  "activeMeetingId": 98373218,
  "activeMotionId": 22325565,
  "isClosing": false,
  "isTie": false,
  "Roles": [],
  "VideoState": 1
}
```

8.3.7. Voting Summary

This is the second option on the menu options. The user will find here the results of each point of discussion/agenda, as well of its details. It is filtered by date and the results can be shown in pages, depending on what the server retrieves as a response.

8.3.7.1 Textual Use Cases

- The voting summary fragment displays a bar on top (under the toolbar), where the user chooses from which date to which date, they want to see the results.

UC4. Check voting summary	
Description	Check the results of previous votes.
Actor	Council member
Preconditions: <ol style="list-style-type: none">1. The user is already logged.2. The device has internet connection.	
Steps: <ol style="list-style-type: none">1. The user tabs on the menu button.2. The user selects the voting summary tab.3. The app shows the voting summary activity.4. The user selects the interval or dates or the date, they want to see.5. The app calls the server with the dates given.6. The app shows the information obtained of the server.	
Postcondition: <ol style="list-style-type: none">1. The information demanded by the user is shown.	

- After that, the results are shown, and the user can select which one to see in detail.

UC10. Check agenda results detail	
Description	The user wants to see the agenda results in detail.
Actor	Council member
Preconditions: <ol style="list-style-type: none">1. The user is already logged.2. The device has internet connection.3. The user is already in the voting summary tab and displaying its results.	
Steps: <ol style="list-style-type: none">1. The user tabs on the agenda result they want to see in detail.2. The agenda result is expanded to show in detail.	
Postcondition: <ol style="list-style-type: none">1. The user sees the agenda result in detail.	

- One difference of the agenda in detail is that the agenda results in detail has the results of the motions held during the point of discussion/agenda selected.

UC13. Check motions results	
Description	The user wants to see the results of a voted motion.
Actor	Council member
Preconditions: <ol style="list-style-type: none"> 1. The user is already logged. 2. The device has internet connection. 3. The user is already in the voting summary tab. 4. The user has already tabbed on one agenda result and it displays its detail. 5. Its detail contains a motion result. 	
Steps: <ol style="list-style-type: none"> 1. The user selects one of the motion results, they want to see. 2. The app opens a motion results dialog. 3. The app displays the requested motion with its results. 	
Postcondition: <ol style="list-style-type: none"> 1. The user sees the results of a voted motion. 	

- Also, the user can check the pdf document attached to the point of discussion/agenda result.

UC12. Read pdf document 2	
Description	The user wants to read a pdf document.
Actor	Council member
Preconditions: <ol style="list-style-type: none"> 1. The user is already logged. 2. The device has internet connection. 3. The user is already in the voting summary tab. 4. The user has already tabbed on one agenda result and it displays its detail . 5. Its detail contains a document. 	
Steps: <ol style="list-style-type: none"> 1. The user selects one document of the motion detail. 2. The app asks for the document to the server. 3. The app opens the document activity. 4. The app displays the requested document. 	
Postcondition: <ol style="list-style-type: none"> 1. The user reads the document. 	

8.3.7.2. REST Requests

- (GET) {SERVER_URL}/votingresults

Get the results of a point of discussion/agendas with its detail, attached documents and the results of the motions held.

Request arguments:

This service has the argument body empty.

Header:

- **x-access-token.** User access token.

Query:

- **String dateFrom.** String of the date from it is wanted to be searched with format yyyy/mm/dd.
- **String dateTo.** String of the date till it is wanted to be searched with format yyyy/mm/dd.
- **Int page.** Current page of the results.

Response:

Body arguments:

- **List<VoteResult> voteResults.** List of the votes results of the session.
A VoteResult has the following attributes:
 - **String abstract.** Explaining text of the voteResult.
 - **Bool AreMeetingMinutesPublished.** Not used by the app.
 - **String assignments.** Assignments of voteResults.
 - **Double DateActivated.** TimeStamp of the date the voteResult was activated.
 - **List<Document> documents.** List of documents contained in voteResult.
 - **String id.** Id of the voteResult.
 - **String name.** Name of the voteResult.
 - **String owner.** Name of the owner of the voteResult.
 - **String requestedBy.** Name of the requester of voteResult
 - **Sponsor.** Sponsor of the voteResult
 - **List<Motion> motions.** Motions in the voteResult
- **Double totalCount.** Number of voteResults obtained.
- **Double totalPages.** Number of pages obtained.

Example Response

```
{
  "voteResults":[
    {
      "abstract":"Lorem ipsum dolor sit amet, consectetur",
      "AreMeetingMinutesPublished":true,
      "assignments":"Assignment 1/nAssignment 2",
      "DateActivated":318198607000,
      "documents":[

      ],
      "id":"6155",
      "name":"2015-1494-001",
      "owner":"123-456, Acme Corp.",
      "requestedBy":"Runner, Road",
      "Sponsors":"Council Member A, Council Member B",
      "motions":[
        {
          "description":"lorem",
          "id":123,
          "movedBy":"Council Member 87",
          "notVoting":true,
          "result":"YES",
          "rollcalls":[
            "Council Member A",
            "Council Member B"
          ],
          "secondedBy":"President A",
          "sequence":1,
          "voteClosed":true,
          "votingOptionsResult":[
            {
              "Color":"green",
              "id":12312312312,
              "name":"YES",
              "textColor":"white"
            },
            {
              "Color":"red",
              "id":12312312313,
              "name":"NO",
              "textColor":"white"
            },
            {
              "Color":"yellow",
              "id":12312312314,
              "name":"ABSTAIN",
              "textColor":"white"
            }
          ]
        }
      ]
    }
  ],
  "totalCount": 1
  "totalPages":1
}
```

8.3.8. Video Display

This is the third tab of the menu option, here the user will see the streaming video that it is being played. The video stream type of the url is the “RTSP” protocol.

8.3.8.1. Textual Use Cases

UC5. Display streaming video	
Description	The user wants to see the streaming video.
Actor	Council member
Preconditions: <ol style="list-style-type: none">1. The user is already logged.2. The device has internet connection.	
Steps: <ol style="list-style-type: none">1. The user tabs on the menu button.2. The user selects the display tab.3. The app obtains the streaming video through the server.4. The app shows the video.	
Alternative flow 2.1. There is no streaming video. 2.1.1. The app shows an empty placeholder.	
Postcondition: <ol style="list-style-type: none">1. The user can watch the streaming video.	

8.3.8.2. REST Requests

- **(GET) {SERVER_URL}/chamber/video/stream/configuration**

Get the video Url that is being stream.

Request arguments:

This service has the argument body empty.

Header:

- x-access-token. User access token.

Body arguments:

- **String videoStreamType.** Protocol of the video.
- **String videoStreamURL** Url of the video to display.

Example:

```
{  
    "videoStreamType": "rtsp",  
    "videoStreamURL": "rtsp://wowzaec2demo.streamlock.net/vod/mp4:BigBuckBunny_115k.mov"  
}
```

8.3.9. Request to speak

8.3.9.1 Textual Use Cases

- The user places a request to speak in the chamber.

UC7. Request to speak	
Description	Send a petition to speak.
Actor	Council member
Preconditions: <ol style="list-style-type: none">1. The user is already logged.2. The device has internet connection.	
Steps: <ol style="list-style-type: none">1. The user tabs the request to speak button.2. The app sends a signal through the server to request to speak.3. The app changes the request to speak button label to "REQUEST TO SPEAK PLACED".4. The president accepts the request to speak and the user can start speaking (UC9).	
Alternative flow 4.1. The president declines the speak request. 4.1.2. The app changes the button label again to "REQUEST TO SPEAK".	
Postcondition: <ol style="list-style-type: none">1. The user has successfully made a request to speak.	

- The user has already been approved to speak.

UC11. Speak	
Description	The user wants to see one point of discussion in detail.
Actor	Council member
Preconditions: <ol style="list-style-type: none">1. The user is already logged.2. The device has internet connection.3. The user has already made a request to speak.4. The request to speak made, it has been accepted.	
Steps: <ol style="list-style-type: none">1. The request to speak button label changes to "SPEAKING" and its color to green.2. The user speaks.3. When the time to speak ends, the label of the button changes again to "REQUEST TO SPEAK".	
Alternative flow 2.1. The president mutes their microphone. 2.1.2. The request to speak button label changes its label to "MUTED" and its color to red.	
Postcondition: <ol style="list-style-type: none">1. The user has spoken during their turn.	

8.3.9.2. REST Requests

- **(POST) {SERVER_URL}/requestToSpeak**

Send a request to speak.

Request arguments:

This service has the argument body empty.

Header:

- x-access-token. User access token.

Response:

The response is empty

- **(DELETE) {SERVER_URL}/requestTpSpeak**

Remove a request to speak.

Request arguments:

This service has the argument body empty.

Header:

- x-access-token. User access token.

Response:

The response is empty

- **(GET){SERVER_URL}/my/microphone/state**

Check the status of the microphone

Request arguments:

This service has the argument body empty.

Header:

Response:

Body arguments:

- **Int microphoneButtonState.** Defines the status of the microphone.

Example:

```
{  
  "microphoneButtonState": 1  
}
```

9. Conclusions and future work

Technology is increasingly present in the lives of people, and politics could not be an exception. More and more democracies, are feeling the urge to apply the technology to ensure a clean, transparent and right democracy. eParliament was born from that need, trying to simplify the management of the everyday's workflow of a parliamentary chamber.

That solution allows an easy and replicable way to digitize any parliamentary chamber around the world, making it easy to export to any democracy that feels the need to renew their in-chamber tools that make possible a democratic discussion.

Before starting the project, it has been done a lot of research, regarding the certificates and how to success migrating an already web application to an Android environment.

The next step was to make evident the use cases extracted from the specifications the subject required. To do so, the use cases diagram and the textual use cases were defined.

Once the work to do was planned, the UI was designed, trying to make it as similar as the respective web application, trying to optimize it to fit the Android standards.

Consequently, the server-app connection had to be made. The connections were structured to be able to fit the HTTPS protocol the API required. Integrating, this way, all the app services the app needed to work as expected.

Finally, the overall modules of the project, were put together. This way, the app services were combined with the user interface and the data model to provide the user experience expected.

During all the process, the web app developer was contacted in order to have a guidance to follow the web app standards. Thanks to this, the app and the web app can work standalone, using the same back-end.

For now, eParliament, is already working as, its initial release, but there are more features currently being worked on:

- **Device authentication.** .p12 certificate will be removed and a unique identifier of the android devices will be used to validate the authorship of the petitions.
- **Add president role.** Modify the interface of the app to add new features focused on the president of the chamber.

10. References

- [1] Activity component: <https://developer.android.com/guide/components/fundamentals.html>
- [2] Activity lifecycle: <https://developer.android.com/guide/components/activities/activity-lifecycle.html>
- [3] Android Fragments: <https://developer.android.com/guide/components/fragments.html>
- [4] Android Services: <https://developer.android.com/guide/components/services.html>
- [5] Android Data storage: <https://developer.android.com/guide/topics/data/data-storage.html#pref>
- [6] HTTPs definition: <https://www.instantssl.com/ssl-certificate-products/https.html>
- [7] JSON: <https://www.json.org/json-en.html>
- [8] SignalR: <http://signalr.net/>
- [9] Retrofit2: <https://code.tutsplus.com/tutorials/getting-started-with-retrofit-2--cms-27792>