Reverse Engineering Report

Elaborated by:

Daniel Baptista Andrade - 93313

José Luís Rodrigues Costa - 92996

March 29, 2022

Summary

The application UAMobile, the app in study in this report, is an app that provides useful information to members enrolled in the University of Aveiro, students, teachers and other uncharacterized members.

UAMobile was developed using a set of educational tools existing in the Moofwd platform and resulted from a partnership between the University of Aveiro and Universia.

For Students, the first screen shows information regarding grades of courses, schedules, room locations, user information and even canteen menus. There are also other buttons that redirect to news of the university itself and its social media, Figure 2.

For teachers, this panel has less buttons, being the present ones the following: Courses, Schedule, Grades, Social Media, News, Notifications Center, Room search, menus, Parking Lots, My data and Settings. A screenshot of this menu is provided later in this report, figure 9.

The third alternative found was the staff menu. This screen has even fewer buttons than the previous one. News, Social Media, Notification Center, Room Search, Menus, Parking Lots, My Data and Settings. From this screen, the three buttons that appear in all the previous menus in the top right of the screen are even missing, as shown below, figure 1.


universidade
de aveiro
um campus que sente
1973 . 2017
Notícias
Redes Sociais
Centro de Notificações
Procura de Salas
Ementas
Parques Estacionamento
Meus Dados
Definições

Figure 1

Overview

This report was elaborated to specify and explain in detail how the android application UAMobile works internally.

By applying reverse engineering concepts it's not only possible to describe the application's functionality from API communication to code structure, but also enumerate all dependencies, permissions and bugs with many useful resources.

Some good practices and recommendations will also be explained in this report in order to mitigate possible application vulnerabilities that have been found alongside the investigation and disassembly of the UAMobile internal components, this includes, for example, not used menus, references to the Porto's University mobile application etc.

The Report

Functionality List

In this subsection, it is provided an overview of the main aspects present in the UAMobile application.

From a regular mobile application user perspective, the functionalities that stand out immediately are the following:

Functionalities	**Description**	**Role**
Login using token or credentials	The app provides a login method in order to use subsequent application functionalities	All users enrolled in Universidade de Aveiro
Show enrolled courses	Screen to visualize which courses the student is enrolled in, including information about classes, schedules, participants etc.	Student, Professor
Show weekly schedule	Screen to visualize the weekly schedule regarding classes	Student, Professor,
Exams	Panel to visualize ECTs per semester as well as the room where the exams are going to take place	Student
Grades history	Screen that shows past grades from the current year	Student
Grades	Panel that shows all the grades since the university enrollment	Student, Professor
Tuition fees	Shows which tuitions are paid and which are not, as well as the value of the non	Student

	paid ones	
Requirements	Requests made in the PACO platform regarding some bureaucracy	Student
News	Latest news in UA platforms	All
Social Media	Facebook and Twitter of UA	All
Social Services Ticket	Tickets for Social Services	Student
Notifications Center	Notifications related to which functionality present in this table	All
Room search	Screen that shows where a given room is located	All
Menus	Panel to visualize the menus from UA canteens	All
Parking Lots	Screen showing all the UA parking lots and their availability	All
My Data	Personal information of the logged user	All
Settings	Menu with information about the app, help and some small options to tweak notifications	All
Show all menu options	Menu used to select functionalities, different for each user type	All

Note: All - Student, Professor and Staff

Some of the functionalities appear to be a bit “buggy” and some of the menus can be misleading or confusing, for example the Grades and the Grades history menus. Using the app on a day to day basis as a Universidade de Aveiro student, the “Requirements” screen did not have any functionality, because it could be dependent from an online platform such as PACO.

Below, it is possible to see the main screen of the app from a student perspective.

Figure 2

Dependencies and versions

In the Android manifest of the application it is possible to note that the android minSDK Version is “14” and it is built to run on the target version of “26”.

After an overall look inside the disassembled application, it is possible to note a folder called “sources'' which contains a “com” and a “org” folder. The first one contains the directory responsible for storing some dependencies, as well as, the source code of the app. With this information, we can enumerate some of them:

- com.bumptech.glide (Glide Bumptech) v3.6.0/3.6.1 : An image loading and caching library for Android focused on smooth scrolling

- com.google.android.gms(GMS) : Responsible for providing APIs and application for android devices

- com.squareup.picasso (Picasso) v2.0 : Responsible for providing and caching images for Android applications

- org.jsoup (Jsoup) v1.12 : A Java library responsible to create interaction between Java and HTML

As there is no easy method to identify the library versions, by comparing with the github version history and verifying the changes on each release, it’s possible to identify the approximate version.

Even though code analysis shows that only Picasso dependency is not used, all of these libraries are outdated and should be updated to diminish any exploitation possibility.

Application Permissions

This application declares the following permissions on the android manifest file:

- permission.INTERNET: Allows applications to open network sockets
- permission.READ_PHONE_STATE: Allows read access to the device's phone number(s).
- permission.ACCESS_NETWORK_STATE: Allows applications to access information about networks.
- permission.WRITE_EXTERNAL_STORAGE: Allows an application to write to external storage.
- permission.CALL_PHONE: Allows an application to initiate a phone call without going through the Dialer user interface.
- permission.VIBRATE: Allows access to the vibrator
- permission.GET_ACCOUNTS: Allows access to the list of accounts in the Accounts Service.
- permission.WAKE_LOCK: Allows using PowerManager to keep the processor from sleeping.
- com.google.android.c2dm.permission.RECEIVE: Grants permission to receive messages from Google Cloud Messaging.
- android.permission.READ_EXTERNAL_STORAGE: Allows an application to read from external storage.
- android.permission.USE_CREDENTIALS - Allows the app to request authentication tokens.

From a reverse engineering point of view, identifying all permissions helps establish a correlation between the permissions and the potential app functionality.

Considering all application functionality explained in the chapter *“Functionality List”,* some permissions stand out as unnecessary. For example the permission.CALL_PHONE is defined, but there is no direct or indirect functionality in the app that allows the device to perform a call.
The definition of unnecessary permissions may indicate that security best practices were not taken into account in the applications development.

Application WorkFlow

To better understand the application activity structure this section of the report will focus on all information about the activities of the application, detailing specific API requests, which functionalities are implemented, which information stored and updated and some relevant annotations for each activity.
The functionality list of each activity is directly correlated with the whole application functionality list mentioned in the section *“Functionality List”.*

Activity #1 - Home Activity

User: Student

3 Entry points:

- SplashScreen
- Login
- Settings

19 Exit points:

- Message
- Settings
- Event
- Notification
- Parking
- Announcement
- Schedule
- HistoricalCourse
- SearchClassroom
- Finance
- Course
- Food
- Grades
- Social
- Moche
- Petition
- News
- Exam
- Ticket

Functionality list:

- Show all menu options

API calls:

- DailyLogin Request:

```
“service”:”authLoginLightClient”,
“params”: { “userToken”:”xxxx”,
            “roleId”:”xxxx”,
            “wsToken”:”xxxx”,
            “clientVersion”:”2.0.4”,
            “userId” : “xxxx”,
            “deviceId” : “xxxx”,
          “username” : “xxxx”
}}
```

Additional Information:

- The Home screen will vary with user type, the professor menu only has 14 options and the staff one 8 options.
- The Moche screen is misleading, because it was modified to show news in the student dashboard instead of its original purpose (maybe advertising?)

Activity #2 - Ordinary Activity

1 Entry point:

- Home Activity

0 Exit points

Functionality list:

- Show weekly Schedule
- Exams
- Grades History
- Grades
- Tuition Fees
- Requirements
- News
- Social Media
- Social Services Ticket
- Menus
- Parking Lots
- My Data

API calls:

Weekly Schedule:

service: scheduleGetScheduleClient
params: {
userData:”xxxx”,
roleId:”xxxx”,
wsToken:”xxxx”,
lang: “pt”
}}

Exams:

service: examsGetExamsClient
params: {
roleId:”xxxx”,
userNC:”xxxx”,
token:”xxxx”
}}

Grades History:

service: coursesHistoricalListClient
params: {
listToDisplay:”xxxx”,
roleId:”xxxx”,
userNC:””,
token:””
}}

Grades:

service: scheduleGetScheduleClient
params: {
userData:”xxxx”,
roleId:”xxxx”,
wsToken:”xxxx”,
lang: “pt”
}}

Tuition Fees:

service: financesGetListStudentClient
params: {
user_id:”xxxx”,
roleId:”xxxx”,
wsToken:”xxxx”,
userNC: “xxx”
}}

Requirements:

service: petitionGetStatusClient
params: {
userNC:”xxxx”,
roleId:”xxxx”,
token:”xxxx”,
lang: “pt”
}}

News:

service: tidingsGetCategoriesClient
params: {
role:”xxxx”,
roleId:”xxxx”,
wsToken:”xxxx”,
username: “xxxx”
}}

Social Media:

service: socialSimpleListClient
params: {
userNC:”xxxx”,
roleId:”xxxx”
}}

Social Services Ticket:

service: ticketLineDetailsClient
params: {
userNC:”xxxx”,
roleId:”xxxx”,
wsToken:”xxxx”
}}

Menus:

service: mealsGetListClient
params: {
role:”xxxx”,
roleId:”xxxx”,
wsToken:”xxxx”,
}}

Parking Lots:

service: parkingDetailsClient
params: {
roleId: “xxxx”
userData:”xxxx”,
userToken:”xxxx”,
wsToken:”xxxx”,
userNC: “xxxx”,
userId: “xxxx”, lang:

My Data:

service: profileGetPrivateClient
params: {
userToken:”xxxx”,
userData: “xxxx”,
roleId:”xxxx”,
wsToken:”xxxx”,
userData:“xxxx”,

“pt” }}	lang: “pt” }}

Additional Information:

- All these screens are very similar in terms of how they request and process the information, therefore they are grouped inside this activity
- Some of them have “exit” points, but those new menus have no relevant information that is not present in the prior screen
- Some of the requests deviate from the common request structure, mainly by using a “token” field that contains information normally named as “wsToken”. This indicates a specific API server side implementation for some services.

Activity #3 - Courses Activity

1 Entry point:

- Home Activity

4 Exit points:

- Schedule
- Participants
- Classes
- Tasks

Functionality list:

- Show enrolled courses

API calls:

- Get Courses Request:

```
{“service”:”courseGetListClient”,
 “params”: { “userToken”:”xxxx”,
            “roleId”:”xxxx”,
            “wsToken”:”xxxx”,
            “userData”:”xxxx”,
            “userNC” : “xxxx”,
            “lang” : “pt”,
           “userId” : “xxxx”
}}
```

Additional Information:

- This screen provides a schedule of the classes, the students enrolled, the number of classes that have been lectured and tasks

Activity #4 - Notification Center Activity

1 Entry point:

- Home Activity

0 Exit points

Functionality list:

- Notification Center

API calls:

Additional Information:

- This screen provides a list of notifications regarding all the activities present in the Home Activity
- The Chat submenu seems to be the most needed, since the button that appears in the Home Activity does not function properly and it is the only way of seeing older messages from previous courses

Activity #5 - Room Search Activity

1 Entry point:

- Home Activity

2 Exit points:

- See Building
- List Rooms

Functionality list:

- Room Search

API calls:

- Search room Request:

```
{“service”:”locationClassRoomGetListClient”,
 “params”: { “userToken”:”xxxx”,
            “roleId”:”xxxx”,
           “wsToken”:”xxxx”,
           “userNC”:”xxxx”,
           “userId” : “xxxx”,
           “room” : “xxxx”,
           “lang” : “pt”
}}
```

Additional Information:

- Sometimes if the room has a single digit, like 4.1.2, all the 20s room appear in a list
- This activity uses a temporary file called “SearchClassroom.fwd” to store temporary information about the last room search.

Activity #6 - Settings Activity

1 Entry point:

- Home Activity

3 Exit points:

- Notifications
- Usage
- About
- Help

Functionality list:

- Settings

API calls:

- Logout Request:

```
{“service”:”authLogoutClient”,
 “params”: { “wsToken”:”xxxx”
}}
```

- Get Notification Preferences Request:

```
{“service”:”notificationGetCategories”,
 “params”: { “roleId”:”xxxx”,
             “userNC”:”xxxx”,
             “lang”:”pt”
}}
```

Additional Information:

- The list below the “Information” tag open a text explaining either how the app works (Help) or the context of why the app was built(About)
- Notifications and Usage are submenus with the ability of tweaking parameters related to notification popups above the fragments on the home screen. This information is based on how the code should work, none of the fragments seem to suffer any change after all.

Data Persistence

This application mainly uses the SharedPreferences API provided by the android operating system to store data. This API allows the application to save and retrieve data in the form of a key value pair, storing this information in a clear text xml located in “/data/data/your.app.id/shared_prefs”. This path can be accessed either with rooted devices or via a debuggable android app.
In this application, specifically the shared preferences, is used to store the following data about authentication, application state and user data:

Authentication:

- USER_TOKEN : User Token
- USERNAME : User identifier
- PASSWORD : Always blank
- WS_TOKEN : Authentication Token
- URL : SSO authentication url

Application State:

- LOGIN : If the user is logged in
- IS_MULTIPROFILE : If the user has access to multiple profiles
- IS_REGISTER : If the user is registered to notifications
- NUM_MAX_LIST : Maximum number of items in a list
- DAILY_LOGIN : Latest login
- DATE_SYNC_NOTIFICATIONS : Sync notifications

User data:

- USER_NC : Email
- ROLE_ID : Student, Professor, Staff

The figure 3 illustrates an example of the shared preferences file previously mentioned.

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="role">Student</string>
    <long name="secondsToChangeBackground" value="3600000" />
    <long name="secondsToCheckDailyLogin" value="86400000000" />
    <string name="privacy">https://www.ua.pt/termos_de_utilizacao</string>
    <boolean name="login" value="true" />
    <string name="library">http://www.moofwd.com</string>
    <string name="portal">https://www.ua.pt/</string>
    <string name="contactUs">https://www.ua.pt/Contacts.aspx</string>
    <long name="secondsToRefreshList" value="1800000" />
    <long name="timeToSyncDeviceConfig" value="1649780913409" />
    <long name="dateSyncNotifications" value="1649780913239" />
    <string name="roleId">2</string>
    <string name="profile">student</string>
    <string name="userFullName">JOSÉ LUÍS RODRIGUES COSTA</string>
    <boolean name="isMultiprofile" value="false" />
    <string name="wsToken"></string>
    <string name="userNC">joselcosta@ua.pt</string>
    <string name="userId">joselcosta@ua.pt</string>
    <string name="url"></string>
    <boolean name="firstTime" value="false" />
    <string name="userToken"></string>
    <string name="user_id">joselcosta@ua.pt</string>
    <long name="secondsToSyncNotifications" value="1800000" />
    <long name="secondsToSyncDeviceConfig" value="1800000" />
    <boolean name="isRegister" value="true" />
    <string name="username"></string>
</map>
```

Figure 3

As information stored via shared preferences is stored in local storage in clear text, it’s not recommended to store critical information such as the WS_TOKEN . When this information must be stored in this manner, it should be encrypted.

UAMobile API Communication Structure

This application uses a fixed structure that allows the multiple activities with functionalities to communicate with the external API as shown in figure 4.

It uses multiple Java classes that extend from an abstract class, MoofwdTask, one for each activity of the application with specific requests for all the different functionalities of the activity.
These are easily recognizable since all of them end with “Task”.

The MoofwdTask contains important functions such as the executeTask() that must be implemented specifically by each specific extended class, that creates the request that will be sent to the external API using the callMashup() function and when the response arrives, the mashupFinishedWithResponse() function will be called asynchronously to handle the received information by creating a JSON object that is stores the server’s response.

To perform this API call with callMashup(),the MoofwdTask class internally uses a ZubronAPI_1_0 instance that is responsible for granting or denying access to the API, according to its parameters, calling the mashup() to perform the API call.

This ZubronAPI_1_0 instance also contains a ZubronMashup_1_0 asyncTask that is responsible for the async HTTP request to the external API. It extends a normal native Android asyncTask, and implements the onPostExecute() function which sets up the request, and starts the AsyncTask that performs the doInBackground() function in a separate thread. This function sends the POST request.

This structure allows for an easy addition of new functionalities to the application, as only a new Moofwd Task has to be created with the proper request and response data handling.

Both ZubronAPI_1_0 and ZubronMashup_1_0 belong to a different package from the whole application, which indicates it most likely is reused in multiple applications that communicate with the same external API implementation.

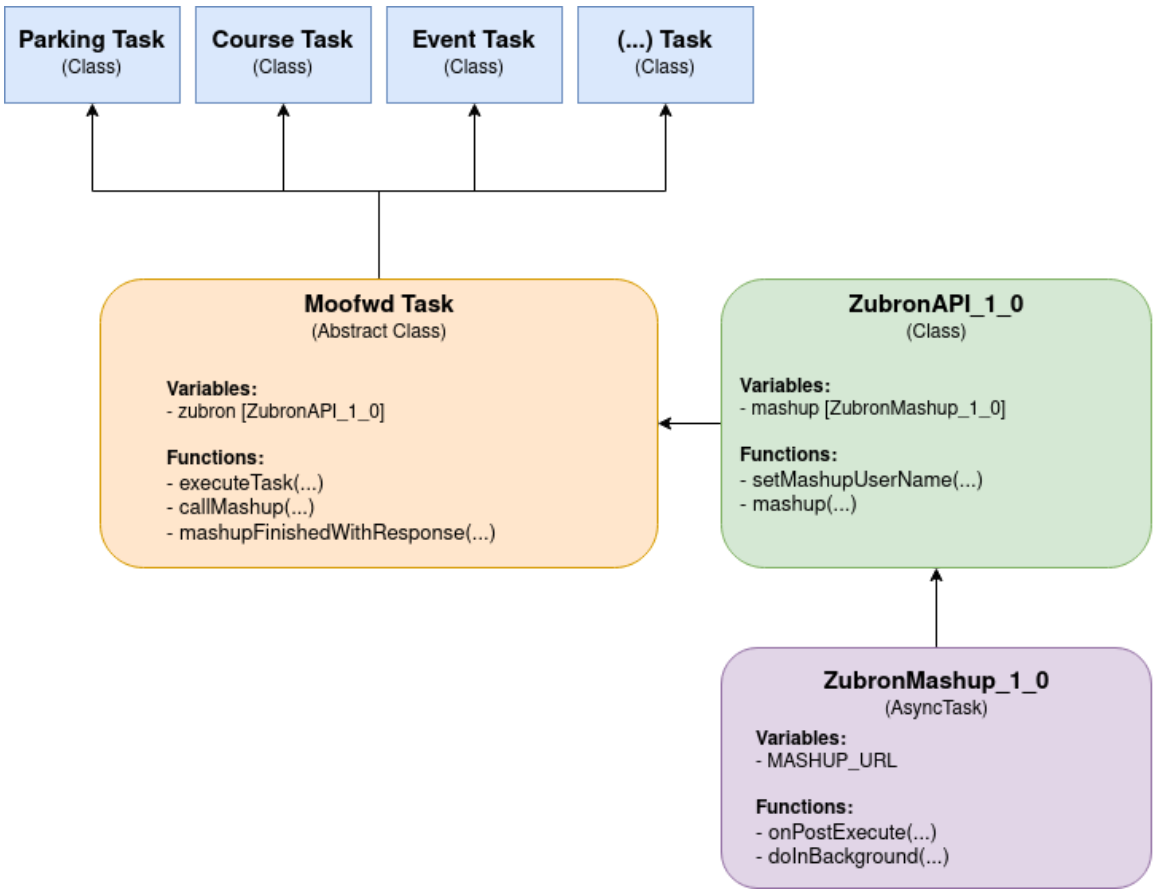

Parking Task
(Class)
Course Task
(Class)
Event Task
(Class)
(...) Task
(Class)
Moofwd Task
(Abstract Class)
Variables:
- zubron [ZubronAPI_1_0]
Functions:
- executeTask(...)
- callMashup(...)
- mashupFinishedWithResponse(...)
ZubronAPI_1_0
(Class)
Variables:
- mashup [ZubronMashup_1_0]
Functions:
- setMashupUserName(...)
- mashup(...)
ZubronMashup_1_0
(AsyncTask)
Variables:
- MASHUP_URL
Functions:
- onPostExecute(...)
- doInBackground(...)

Figure 4

API characterization

In this section of the report, the external API will be explained in more detail, this includes all information about the exchanged messages according to the multiple functionalities of the application, the message structure that is exchanged and the (possible) meaning of all request fields.
This application mainly communicates with the following endpoint:

- uaverio2.mooestro.com/moofwd-rt/gateway.sjson

This endpoint is used to request all information that allows the application functionalities to execute correctly.

All application calls to this endpoint use the HTTP Post method, this behavior does not follow conventional request methods, such as GET for information data requests and POST for submitting data.

As referenced in this report, this application uses the ZubronAPI package to communicate with this endpoint as explained in the *"UAMobile API Communication"*

All requests to this endpoint generally have a typical structure illustrated in figure 5:

```
POST /moofwd-rt/gateway.sjson HTTP/1.1
"Accept-Encoding":"gzip","deflate"
"Content-Length":"xxx"
"Content-Type":"application/x-www-form-urlencoded"
"Host":"uaverio2.mooestro.com"
"Connection":"close"
"User-Agent": "Apache-HttpClient/UNAVAILABLE (java 1.4)"

json={
  "methodName":"invoke",
  "service":"xxx",
  "appId":"1896194399038728",
  "token":"xxx",
  "params":{
     "xxx":"yyy"
  },
  "clientID":"2",
  "clientVersion":"1.0"
}
```

Figure 5

The typical request structure is a POST request to the /moofwd-rt/gateway.sjson endpoint with a json object that contains information about what information needed to perform the request.
For all requests the structure is maintained, except for the params entry, which will change for each specific request of the application, which may include more fields.

The following table details each parameter of the request:

Parameter	Value	Description
methodName	“invoke”	This parameter is fixed.
service	For each activity, the service is specified in the Application WorkFlow section.	This parameter is used to select which information is going to be requested from the server.
appId	“1896194399038728”	There is no information regarding this parameter, but it’s suspected to be a server side application identifier, as many applications may communicate with the same server.
token	Encrypted with the following form: “*UNS*<username>*UNE**NSS*Moofwd*NSE**TSS*<utctime>*TSE*”	This token parameter is generated by encrypting the string presented in the value section using AES/CBC/PKCS5Padding encryption with the fixed key “2VZ1NG02FX8R8XEC” This token is generated for each request, normally changing the <utctime> field to the request time.
params	For each activity, the params are specified in the Application WorkFlow section.	The params parameter is a json object that contains different fields needed for a successful request according to the service field.
clientID	“2”	This parameter is fixed.
clientVersion	“1.0”	This parameter is fixed.

As explained, the params parameter is a json object used to specify information required to perform a specific service request.
The following table presents the most relevant params options and details each one:

Parameter	Value	Description
userNC	User email	Specifies the user nickname
wsToken	User authentication token	This token uniquely identifies the logged user, is validated by the server and is used for every request, except during the initial phase of the authentication process.
roleId	1, 2, 3, 4	This field indicates the type of user, either professor (1), student (2), multiprofile (3), staff (4).
userToken	XXXX[-M003estr@-]XXX@ua.pt[-M003estr@-]novalue[-M003estr@-]X	Some kind of specific application user identifier
lang	pt, en	Language of the application
userData		Always empty
deviceId	XX:XX:XX:XX:XX:XX	The MAC address of the device wireless network interface

From all of these request fields, by modifying the request parameters, only the wsToken and rodeId are validated by the server and cannot be altered, as the request will be invalidated for most request types.

Authentication

This application uses an authentication API provided by the moofwd gateway and the Aveiro university SSO.
To explain the full authentication process, four diagrams (1, 2, 3 and 4) were created. The diagram 1 illustrates the internal application steps that are executed during the authentication process. The other three diagrams illustrate an overview of the communications between the remote authentication server api and the application.
The authentication process may occur in three different situations illustrated in diagram 1:

- State 1: The user launches the application for the first time and is not logged in
- State 2: The user launches the application, has previously been authenticated and the token is valid
- State 3: The user launches the application, has previously been authenticated and the token is not valid

When the application is in the first state (1) and the user presses the login button located in the Login Activity, the application will start the login process with credentials, communicating with remote server's as illustrated in diagram 2 (Login with credentials):

- Step 1 : The application sends a request to the endpoint requesting a wsToken and url that can be used to validate this token on the Aveiro University SSO and other device parameters.
- Step 2 : The application stores the received information, including the wsToken and the url, opens a Web View and starts a series of redirects until the Aveiro University login page is loaded (figure 6)
- Step 3 : The user either authenticates with the university credentials or with the Portuguese Citizen Card (Chave móvel digital).
- Step 4: The user is authenticated via the OAUTH 2.0 industry-standard protocol.

When the user is authenticated via the Aveiro University login page, the wsToken is validated and can start to be used on the requests.

After all these steps, the user should have a valid wsToken, and the Login Process with token starts.
This process is responsible for the verification of the wsToken validity and requesting all information about the user to be displayed on the Home activity of the application (figure 2). To achieve this, it performs the following communication with the remote server:

- Step 1: The application requests the “AuthLoginLightAuthClient” service to verify the wsToken validity.
- Step 2: After the verification is completed, it requests the “AuthLoginClient” service and provides the response information with user info to the Home activity.

When the application is in the second state (2) the login activity will instantly try to authenticate the user with the valid, available wsToken as the diagram 1 illustrates. After the Login Process with token will start exactly as explained in the first state (1) after obtaining a valid wsToken.

When the application is in the third state (3) the login activity will instantly try to authenticate the user with the invalid, available wsToken as the diagram 1 illustrates. This validation will fail, and the application will start the Login Process with credentials exactly as explained in the first state (1) when the wsToken is not yet available.

Lastly when the application is launched and the user was logged in, the application will start by default in the Home Activity and perform the daily login function when a specific time has elapsed (“secondsToCheckDailyLogin”). The request is illustrated in the diagram 5:

- Step 1: The application requests the “AuthLoginLightAuthClient” service to verify the wsToken validity.

If the wsToken is valid, the application will stay in the Home activity, else it will redirect the user to the Login activity, erasing all user information from SharedPreferences.

All of these authentication steps are shared between all user types of the application.
A clear improvement could be done on this authentication process, when the dailyLogin function verifies that the wsToken is invalid, instead of redirecting back to the login activity, it could start the authentication process immediately.

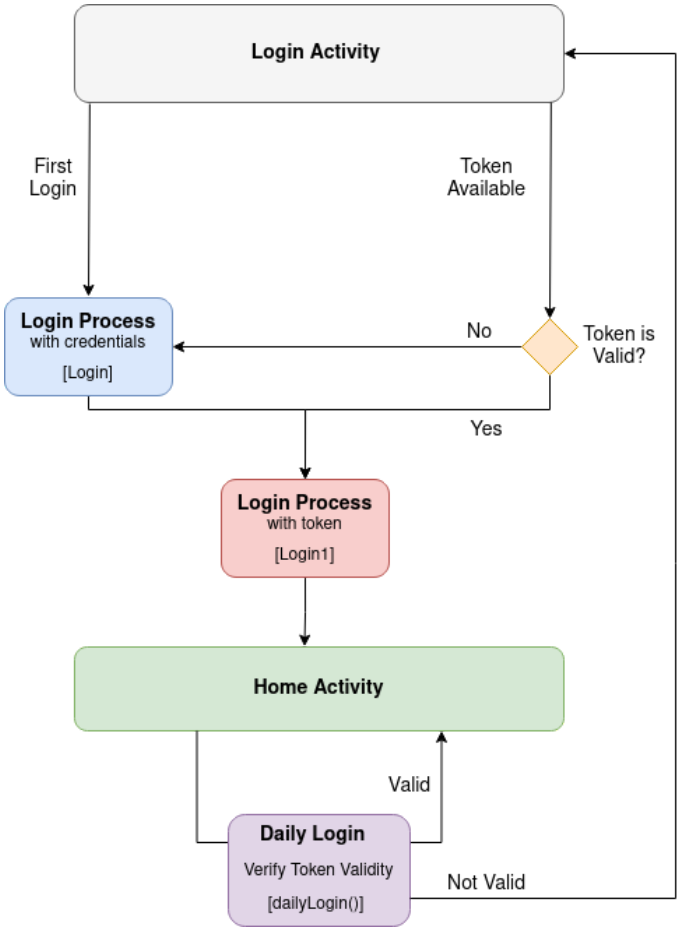
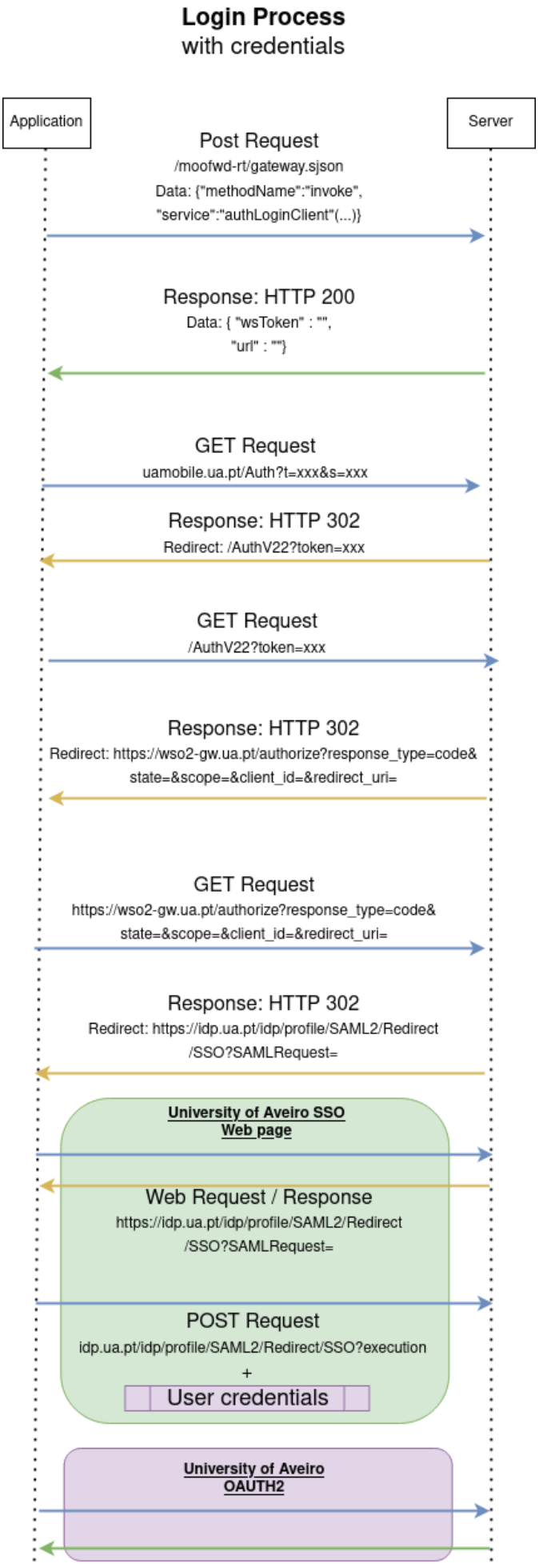
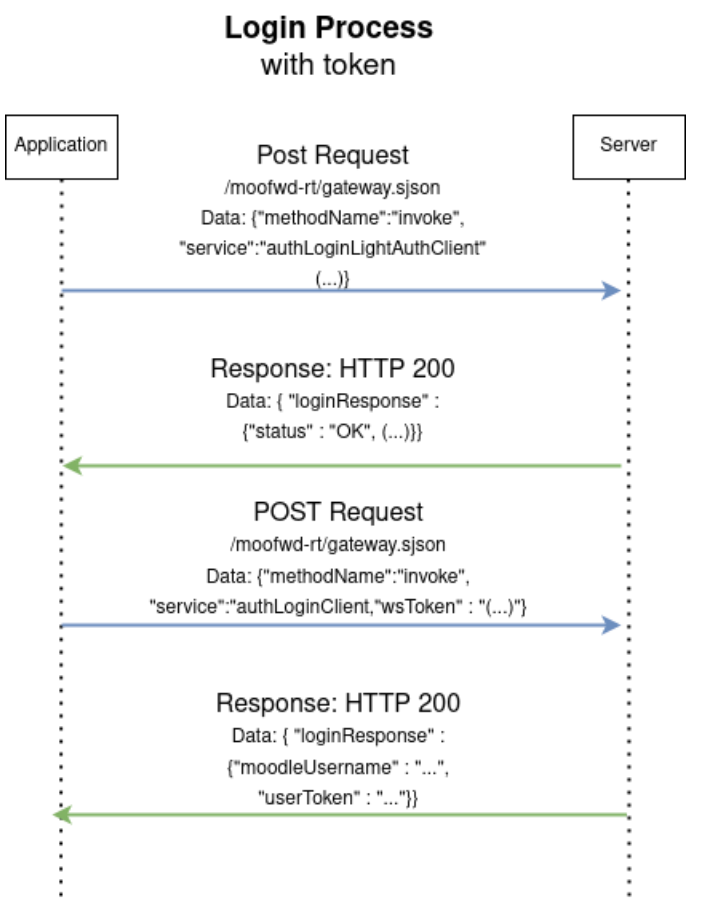
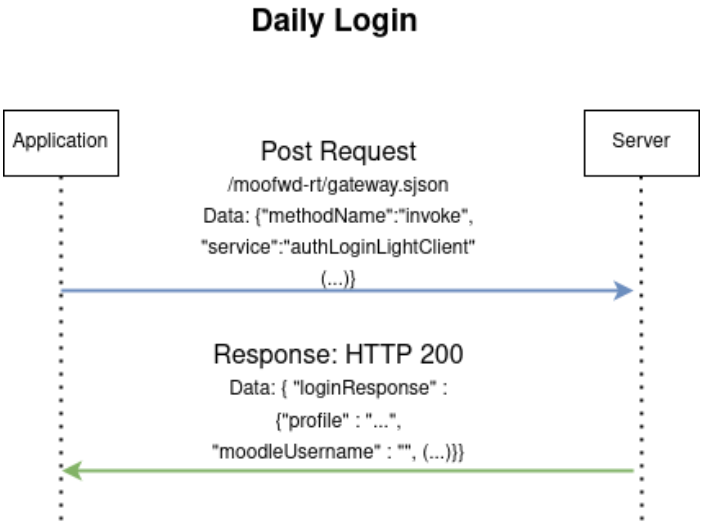


Diagram 1

Figure 6







Diagrams 3, 4 and 5

Other application characteristics

In terms of other characteristics that are non relevant from an user experience perspective, there are some aspects that are interesting and worth mentioning in this subsection.

From file searching, files unrelated to the Universidade de Aveiro have been found. For example, there are assets that are never used and reference other institutions, such as the Universidade do Porto (Figure 7) .

Figure 7

There are also references in the code for websites not related to the UA.

Following the same principles, there is a list of json files, "*ContactUsEnglish"* and "*ContactUsSpanish",* available in the resources folder, with phone numbers and emails that are not from the University. (Figure 8)

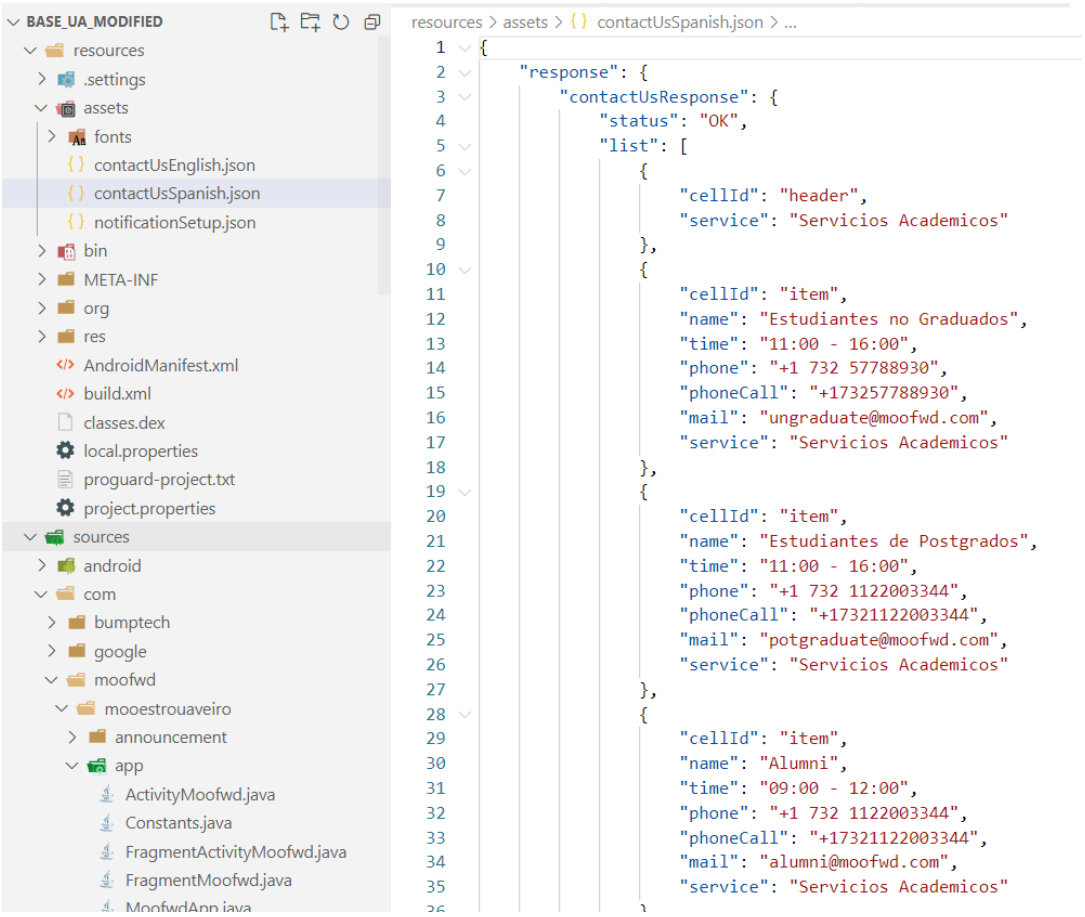


Figure 8

When showing the different menu possibilities for the application, the “professor” menu was obtained by changing the server response or via *frida framework*, a tool that hooks to given methods and changes the flow of the application, as seen in the Figure 9.

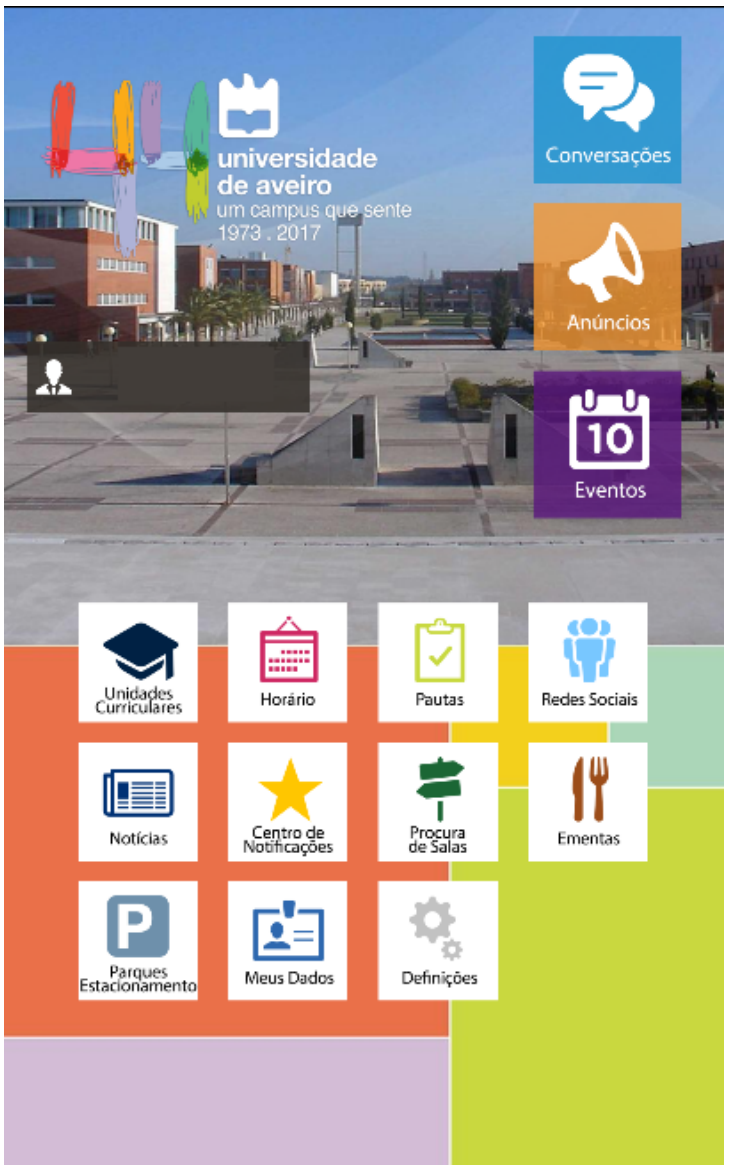


Figure 9

Although this is not very relevant, since the assignment was given, the refresh times as well as the loading ones have become slower than usual. From a regular usage of the app, this can be annoying.

When looking inside the app preferences, a value used to execute the dailylogin function is also present and it is a very large number, equivalent to 1000 days.

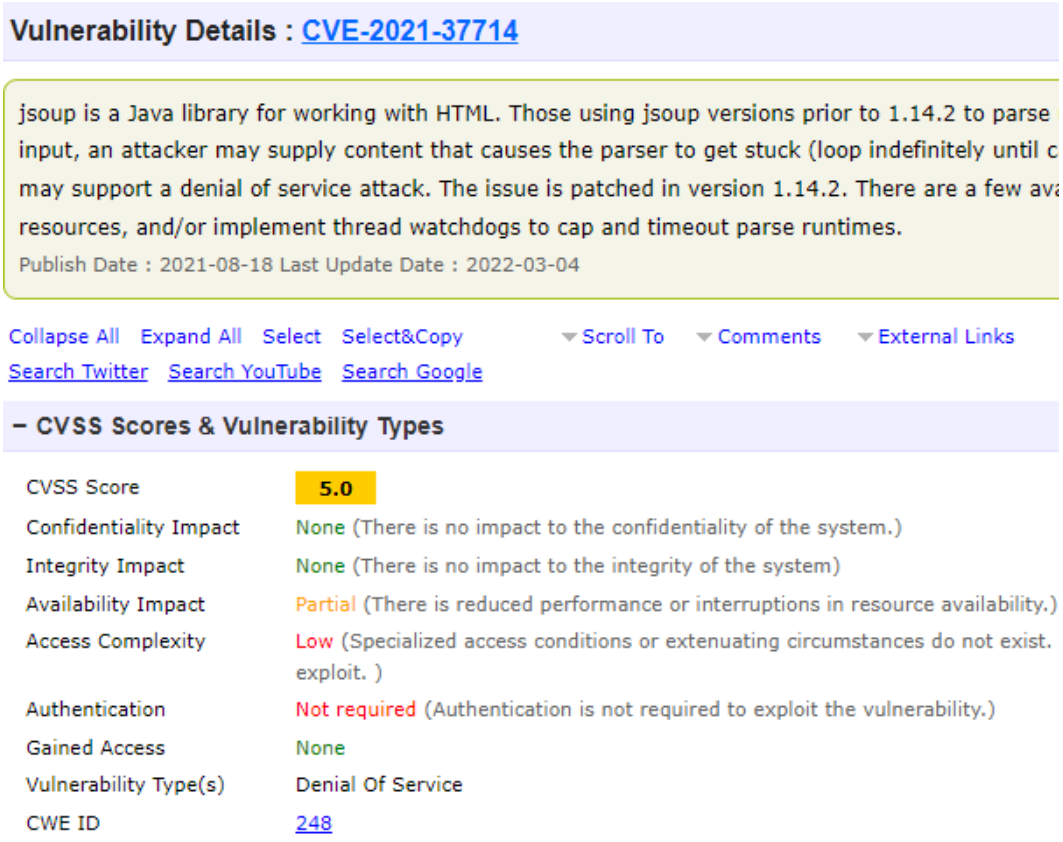
Another characteristic that stood out when reversing this application was the poor coding practices, a great example of this is the inconsistent use of constants when performing requests, this is, using the value of the constant and not the constant itself, which creates inconsistent code.

Bugs and Vulnerabilities

According to the research made to the UAMobile application, there were some bugs and vulnerabilities worth mentioning, which are going to be detailed in the table below:

Bug/Vulnerability	Description
Standard Loading on some screens provide no information	Since the Assignment was given, the standard loading of some screens stops refreshing. It is necessary to log out and log in again to make a clean refresh to the respective screen
Too much detail on the Auth website	Instead of a generic message such as “ Credentials do not match”, the website provides too much detail regarding existing or non-existing users as well as the service endpoint the app is connecting to
Non used permissions	When installing the application and running it for the first time, there are some permissions that need to be granted. However, some of them are not used in the context of the application itself, which can lead to future exploits if the application is compromised in some way
Logs	The developers of the application, for debug purposes, included some logs inside the code. These logs were not removed and expose sensitive information when the correct flag is activated, such as private tokens.
Outdated dependencies	The dependencies declared above on the report are outdated, leading to malfunctions or exploits. One example is the Jsoup version. It is reported in the CVE-2021-37714 that all the versions prior to 1.14.2 suffer from a Denial of Service vulnerability (figure 10).
Profile information in serialized mode	When managing information in the My Data screen, all the provided information is loaded and stored in Serialized form. It is

	known that, for sensitive data, like addresses, serialized contexts in Java must be avoided
Shared preferences not encrypted	Sensitive data like the WS_TOKEN are stored in plaintext in the SharedPreferences



Vulnerability Details : CVE-2021-37714

jsoup is a Java library for working with HTML. Those using jsoup versions prior to 1.14.2 to parse input, an attacker may supply content that causes the parser to get stuck (loop indefinitely until c may support a denial of service attack. The issue is patched in version 1.14.2. There are a few ava resources, and/or implement thread watchdogs to cap and timeout parse runtimes.

Publish Date : 2021-08-18 Last Update Date : 2022-03-04

Collapse All Expand All Select Select&Copy Scroll To Comments External Links
Search Twitter Search YouTube Search Google

– CVSS Scores & Vulnerability Types

CVSS Score	5.0
Confidentiality Impact	None (There is no impact to the confidentiality of the system.)
Integrity Impact	None (There is no impact to the integrity of the system)
Availability Impact	Partial (There is reduced performance or interruptions in resource availability.)
Access Complexity	Low (Specialized access conditions or extenuating circumstances do not exist. exploit.)
Authentication	Not required (Authentication is not required to exploit the vulnerability.)
Gained Access	None
Vulnerability Type(s)	Denial Of Service
CWE ID	248

Figure 10

Conclusion

As the investigation to elaborate this report has been made, several aspects that caught attention have arised.

First, the application itself seemed to be made on top of another already existing application, maybe from Universidade do Porto, due to some assets found as well as some never used code lines. However, even this one seemed to be used from a template, since some numbers and contacts regarding other non-portuguese institutions have been found.

At a first glance, when talking about files and classes not used, immediately a folder named *"Moche"* called attention. Despite not having any partnership with the Universidade de Aveiro, with a little bit of research, it was found that this menu was indeed being used, not in a partnership level, but the classes that constitute that folder had been remade to integrate a "news" button, as seen in the "*DashboardStudentFragment"* class. Although this class has some usage along with the application for student news, assets such as the video of the Universidade do Porto and the real assets from the moche menu as well as the contact list from other places are just useless in the context of this application.

This could indicate that the app was made in sort of a partnership between those institutions mentioned along the folder structure.

When talking about a normal application usage, it is possible to notice a little bit of delay between screens and, sometimes, the screens do not refresh at all being necessary to relogin for some reason. This problem was already mentioned in previous subsections, but found it worth mentioning it as a conclusion point for the report.

For the structure, from accessing endpoints to painting the screens with correct information, the code structure was very consistent, but also misleading when talking about classes that do not operate as their initial intended version.

When talking about the API, Zubron, for every request, this API generates a new token and this could be the main problem when considering the non refresh of the screens such as the Courses one. No evidence on how the server handles these tokens, therefore, some bugs could happen on the other side that the authors of this report cannot see and, for that reason, there

is no mention in this report made regarding how the server operates, just what is the structure of the messages received.

About that, the JSON objects seem to be somewhat consistent, sending the requested information in a very comprehensive way.

The dependencies used seem to be outdated and some CVEs have been found online and referred to in the previous subsection *“Bugs and Vulnerabilities”*. Despite that, in terms of developing an app, there is no need to use two similar libraries to import and cache images for an Android application.

There are some permissions required at the start of the application that are not used at all in the internal processes of the application.

There seems to exist some sort of notification alerts, but none of those notification types popup.

The overall look of the application seems to be outdated, however, it solves the user needs and does what it was built for, displaying information about the University and the user enrolled in it.