Web messaging and notification system between journalists and citizens

Cláudia Margarida da Rocha Marinho
Web messaging and notification system between journalists and citizens

Cláudia Margarida da Rocha Marinho

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Jorge Manuel Gomes Barbosa (Assistant Professor)
External Examiner: Paula Viana (Coordinator Professor)
Supervisor: Maria Teresa Andrade (Assistant Professor)

July 19, 2019
Abstract

Over the recent years, live streaming has become a popular trend among events’ attendees, allowing them to share their own experiences of the event with people around the world. Realizing that this might be an opportunity for content providers to enrich their coverage of large events by making use of the attendees’ contributions, MOG Technologies started a project in a partnership with INESC TEC, Jornal de Notícias and OSTV.

The idea of the project is to offer simple means for the event attendees, who can be general public or professional reporters, to send video streams in real-time to the TV producer who is covering the event and live broadcasting it to the public. The video streams are captured with the smartphones of the attendees through an intuitive web app. On the production side, the intention is to allow the TV event producer to continuously browse through the different received feeds using a drag-and-drop web-based GUI and, at any instant, to select a given feed to be inserted automatically in the main broadcast stream, which can also appear in the on-site large TV screens.

Different challenges were identified in this project, some of which deriving from the necessity of the event producers to communicate with the users of the mobile web app, both citizens and journalists, in order to give them instructions or provide feedback. There is also a need to associate streams collected from the event venue to the corresponding event and to assure the existence of a monetization mechanism on the platform that benefits the content providers.

The present dissertation has been conducted within the context of the aforementioned project with the goal to address the previously identified challenges. Thus, the main objective of this thesis is to specify and develop a system to provide real-time messaging capability between producers and users. In addition, this system should empower the producer to filter live streams by event and to create and distribute advertisements to citizens streaming in an event.

All the goals defined for this system were accomplished successfully and the developed solution was integrated in the main project developed by MOG Technologies. The results of this work were also showcased to the project’s stakeholders that found the new developments to be promising, opening the door to new development opportunities for the product.
Resumo

Nos anos recentes, o fenômeno de live streaming tornou-se uma tendência popular entre os participantes de eventos, permitindo-lhes partilhar as suas experiências do evento com outras pessoas de todo o mundo. Apercebendo-se que esta poderia ser uma oportunidade para que os fornecedores de conteúdo enriquecessem a sua cobertura de eventos de larga escala fazendo uso das contribuições dos participantes, MOG Technologies iniciou um projeto em parceria com INESC TEC, Jornal de Notícias e OSTV.

A ideia do projeto é proporcionar uma forma simples para os participantes de um dado evento, que podem ser cidadãos comuns ou jornalistas, enviarem transmissões de vídeo em tempo real para o produtor de televisão que está a cobrir esse evento e a transmiti-lo ao vivo para o público. As transmissões de vídeo são capturadas com os telemóveis dos participantes através de uma aplicação web intuitiva. Do lado da produção, a intenção é permitir que o produtor do evento examine continuamente as diferentes transmissões recebidas usando uma GUI baseada na web e, em qualquer momento, permitir que este selecione uma determinada transmissão ao vivo para ser inserida automaticamente nos canais de difusão principais, podendo também aparecer nas grandes telas no local do evento.

Diferentes desafios foram identificados neste projeto, sendo que alguns deles originaram-se devido à necessidade dos produtores de eventos comunicarem com os utilizadores da aplicação web móvel, tanto cidadãos como jornalistas, a fim de lhes dar instruções ou fornecer feedback. Também existe a necessidade de associar as transmissões recolhidas do local do evento com o evento correspondente e de assegurar a existência de um mecanismo de monetização na plataforma que beneficia os fornecedores de conteúdo.

A presente dissertação foi realizada no contexto do projeto referido tendo como objetivo a abordagem dos desafios identificados anteriormente. Assim, o principal propósito desta tese é especificar e desenvolver um sistema que fornece um meio de comunicação em tempo real entre os produtores de eventos e utilizadores. Além disso, este sistema deve habilitar o produtor a filtrar transmissões ao vivo por evento e a criar e distribuir anúncios publicitários a cidadãos num evento.

Todos os objetivos definidos para este sistema foram cumpridos com sucesso, sendo que a solução desenvolvida foi efetivamente integrada no projeto principal desenvolvido pela MOG Technologies. Os resultados deste trabalho também foram mostrados aos intervenientes do projeto que acharam os novos desenvolvimentos promissores, abrindo as portas a novas oportunidades de desenvolvimento do produto.
I could not conclude this dissertation without expressing my gratitude to all those who, directly or indirectly, helped me during the course of my dissertation.

First of all, I would like to express my appreciation to my supervisor, Professor Maria Teresa Andrade, for her valuable and constructive suggestions during the planning and development of this dissertation. Her willingness to give her time so generously has been very much appreciated.

Additionally, I would like to acknowledge my co-workers at MOG Technologies not only for their support but also for making my stay at this company much more enjoyable. I would particularly like to single out my supervisor at MOG Technologies, Pedro Santos, to thank him for his patient guidance and useful critiques during the course of this dissertation. I would also like to single out both Alexandre Ulisses and Ivone Amorim for giving me the opportunity to work on this challenge and for their helpful suggestions.

Moreover, I wish to thank my friends and brother for their encouragement and motivation. A special thanks to my fellow finalist, Fabiola Silva, for all the good laughs and for accompanying me on this difficult but rewarding academic journey.

I would also like to acknowledge the support provided by my parents who have always helped me in everything they could. This accomplishment would not have been possible without them.

Lastly, but definitely not least, a very special thanks to Vasco, to whom this work is dedicated, for everything, but especially for being by my side and for providing me with unfailing support while I was working on this thesis.

Cláudia Marinho
“Engineers like to solve problems. If there are no problems handily available, They will create their own problems.”

Scott Adams
# Contents

## 1 Introduction
1.1 Context ........................................ 1
1.2 Project ......................................... 2
1.3 Motivation and Objectives ........................... 3
1.4 Dissertation Structure .............................. 4

## 2 Real-time Web
2.1 Web Basics ....................................... 6
2.2 WebRTC .......................................... 10
  2.2.1 General Architecture ............................ 11
  2.2.2 WebRTC Application ............................ 12
  2.2.3 Signaling Protocols ............................. 15
  2.2.4 Media Server .................................. 16
  2.2.5 Possible Alternatives ........................... 18
2.3 Client-Server Real-Time Web Technologies ............ 19
  2.3.1 Server Push Technologies ....................... 20
  2.3.2 Overview of Server Push Technologies .......... 24
  2.3.3 Socket.IO .................................... 26
2.4 Conclusions ..................................... 27

## 3 Live Streaming Applications
3.1 Consumer Oriented Platforms ......................... 29
  3.1.1 Facebook Live ................................ 30
  3.1.2 Twitch ........................................ 31
  3.1.3 Discord ....................................... 31
3.2 Business to Business (B2B) Platforms ................. 32
  3.2.1 Livestream ................................... 32
  3.2.2 Bambuser ..................................... 34
  3.2.3 DaCast ........................................ 35
3.3 Comparative Analysis ................................ 35
3.4 Conclusions ..................................... 38

## 4 MOGPlay Platform
4.1 Overview ........................................ 39
4.2 Current Solution ................................... 41
4.3 Challenges ...................................... 45
4.4 Problem Definition ................................. 47
4.5 Conclusions ..................................... 48
## Operate & Notify Ecosystem

5 Operate & Notify Ecosystem
5.1 Overview ................................................. 50
5.2 Methodology ............................................. 54
5.3 Requirements Definition ................................. 55
   5.3.1 Functional Requirements ............................. 55
   5.3.2 Non-Functional Requirements ......................... 57
   5.3.3 Use Case ........................................... 58
5.4 Architecture ............................................. 60
   5.4.1 Data Access Layer .................................... 61
   5.4.2 Application Server ................................... 63
   5.4.3 Communication Architecture ......................... 64
   5.4.4 Additional Implementation Details .................... 66
   5.4.5 Deployment .......................................... 67
5.5 Prototype Implementation ............................... 68
   5.5.1 Authentication Module ............................... 70
   5.5.2 Messaging Module ................................... 75
   5.5.3 Platform Management Module ......................... 79
5.6 Results .................................................. 84

6 Conclusions and Future Work ............................. 87
6.1 Future Work ............................................. 89

A Prototype ................................................ 91
A.1 Mobile Web Application ................................. 91
A.2 Web-based GUI ......................................... 95

References ................................................. 99
## List of Figures

1.1 CHIC project’s overall structure ........................................... 2
1.2 General workflow for the MOGPlay project .............................. 3

2.1 Web technologies [10] ....................................................... 8
2.2 Traditional HTTP client-server model .................................... 9
2.3 WebRTC architecture [18] .................................................. 11
2.4 Basic WebRTC application [22] .......................................... 12
2.5 Shared signaling channel [19] ............................................. 13
2.6 WebRTC broadcast with no server, adapted from [26] ............... 16
2.7 Bit rate comparison for WebRTC media servers [29] ................. 18
2.8 HTTP Polling in a web application [37] .................................. 20
2.9 HTTP Long-Polling in a web application [37] .......................... 21
2.10 Server-Sent Events in a web application [39] .......................... 22
2.11 WebSocket protocol in a web application [39] .......................... 23
2.12 Network traffic in HTTP Polling and WebSocket applications [42] 24
2.13 Latency in HTTP Polling and WebSocket applications [42] ...... 25
2.14 Socket.IO in a Node.js application [45] ................................. 27

3.1 Facebook Live’s user interface [47] ....................................... 30
3.2 Livestream Studio’s user interface ........................................ 33

4.1 High-level overview of the MOGPlay project .......................... 40
4.2 Mobile web application by MOG Technologies .......................... 41
4.3 Web-based GUI by MOG Technologies ................................... 42
4.4 Architecture solution for MOGPlay by MOG Technologies ........ 43

5.1 Communication mechanism between event producers and streamers .... 51
5.2 Solution to filtering web streams by event ................................. 52
5.3 Platform monetization solution with example ........................... 53
5.4 Iterative model [51] .......................................................... 54
5.5 Disposition of attendees at event venue .................................. 59
5.6 Logical architecture of Operate&Notify .................................. 60
5.7 Conceptual model ............................................................. 62
5.8 Communication architecture ................................................. 64
5.9 Initial version of the watcher interface .................................... 68
5.10 Streamer page before and after integration with the main project .... 69
5.11 Overview of MOGPlay’s user types ....................................... 70
5.12 Mobile app’s home page .................................................... 71
5.13 Streamer’s page ............................................................... 71
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.14</td>
<td>Difference between authenticated and anonymous users</td>
<td>72</td>
</tr>
<tr>
<td>5.15</td>
<td>General area login</td>
<td>72</td>
</tr>
<tr>
<td>5.16</td>
<td>Journalist area login</td>
<td>72</td>
</tr>
<tr>
<td>5.17</td>
<td>Authentication system on mobile application</td>
<td>73</td>
</tr>
<tr>
<td>5.18</td>
<td>Web-based GUI’s home page</td>
<td>74</td>
</tr>
<tr>
<td>5.19</td>
<td>Journalist registration</td>
<td>74</td>
</tr>
<tr>
<td>5.20</td>
<td>Messaging system in watcher page</td>
<td>75</td>
</tr>
<tr>
<td>5.21</td>
<td>Individual messaging system</td>
<td>76</td>
</tr>
<tr>
<td>5.22</td>
<td>Group audio message</td>
<td>77</td>
</tr>
<tr>
<td>5.23</td>
<td>Producer’s messaging options</td>
<td>77</td>
</tr>
<tr>
<td>5.24</td>
<td>Selection mode (group messages)</td>
<td>78</td>
</tr>
<tr>
<td>5.25</td>
<td>Create event page</td>
<td>79</td>
</tr>
<tr>
<td>5.26</td>
<td>Manage events page</td>
<td>80</td>
</tr>
<tr>
<td>5.27</td>
<td>Create advertisement page</td>
<td>81</td>
</tr>
<tr>
<td>5.28</td>
<td>Event’s advertising campaign information</td>
<td>82</td>
</tr>
<tr>
<td>5.29</td>
<td>Set advertising campaign</td>
<td>82</td>
</tr>
<tr>
<td>5.30</td>
<td>Create group page</td>
<td>84</td>
</tr>
<tr>
<td>A.1</td>
<td>General area registration (citizen registration)</td>
<td>91</td>
</tr>
<tr>
<td>A.2</td>
<td>Journalist login page (permission denied error)</td>
<td>91</td>
</tr>
<tr>
<td>A.3</td>
<td>Citizen login page (login success)</td>
<td>92</td>
</tr>
<tr>
<td>A.4</td>
<td>Streamer page (invalid event error)</td>
<td>92</td>
</tr>
<tr>
<td>A.5</td>
<td>Streamer page (anonymous user)</td>
<td>92</td>
</tr>
<tr>
<td>A.6</td>
<td>Streamer page menu (authenticated journalist)</td>
<td>92</td>
</tr>
<tr>
<td>A.7</td>
<td>Edit profile page</td>
<td>93</td>
</tr>
<tr>
<td>A.8</td>
<td>Dashboard menu</td>
<td>93</td>
</tr>
<tr>
<td>A.9</td>
<td>Delete profile modal</td>
<td>93</td>
</tr>
<tr>
<td>A.10</td>
<td>Change password page</td>
<td>93</td>
</tr>
<tr>
<td>A.11</td>
<td>Streamer page (landscape mode)</td>
<td>94</td>
</tr>
<tr>
<td>A.12</td>
<td>Streamer page (hidden content)</td>
<td>94</td>
</tr>
<tr>
<td>A.13</td>
<td>Watcher page (individual text message)</td>
<td>95</td>
</tr>
<tr>
<td>A.14</td>
<td>Publish stream modal</td>
<td>95</td>
</tr>
<tr>
<td>A.15</td>
<td>Messaging options modal</td>
<td>96</td>
</tr>
<tr>
<td>A.16</td>
<td>Dashboard home page</td>
<td>96</td>
</tr>
<tr>
<td>A.17</td>
<td>Manage advertisements page</td>
<td>97</td>
</tr>
<tr>
<td>A.18</td>
<td>Create group modal</td>
<td>97</td>
</tr>
<tr>
<td>A.19</td>
<td>Manage groups page</td>
<td>98</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Overview of server push technologies, adapted from [35] .......... 26
3.1 Overview of live streaming applications ......................... 36
5.1 Functional requirements for the mobile web application .......... 56
5.2 Functional requirements for the web-based GUI ................. 57
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AJAX</td>
<td>Asynchronous JavaScript And XML</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>B2B</td>
<td>Business to Business</td>
</tr>
<tr>
<td>CDN</td>
<td>Content Delivery Network</td>
</tr>
<tr>
<td>CHIC</td>
<td>Cooperative Holistic View on Internet and Content</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>DTLS</td>
<td>Datagram Transport Layer Security</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HLS</td>
<td>HTTP Live Streaming</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Hypertext Transfer Protocol Secure</td>
</tr>
<tr>
<td>ICE</td>
<td>Interactive Connectivity Establishment</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>MCU</td>
<td>Multipoint Control Unit</td>
</tr>
<tr>
<td>MPEG-DASH</td>
<td>MPEG Dynamic Adaptive Streaming over HTTP</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translator</td>
</tr>
<tr>
<td>NTP</td>
<td>Network Time Protocol</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RTCP</td>
<td>Real-Time Control Protocol</td>
</tr>
<tr>
<td>RTMP</td>
<td>Real-Time Messaging Protocol</td>
</tr>
<tr>
<td>RTP</td>
<td>Real-time Transport Protocol</td>
</tr>
<tr>
<td>SCTP</td>
<td>Stream Control Transmission Protocol</td>
</tr>
<tr>
<td>SDP</td>
<td>Session Description Protocol</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SFU</td>
<td>Selective Forwarding Unit</td>
</tr>
<tr>
<td>SIP</td>
<td>Session Initiation Protocol</td>
</tr>
<tr>
<td>SRTP</td>
<td>Secure Real-time Transport Protocol</td>
</tr>
<tr>
<td>STUN</td>
<td>Session Traversal Utilities for NAT</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TURN</td>
<td>Traversal Using Relays around NAT</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>VoIP</td>
<td>Voice over Internet Protocol</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WebRTC</td>
<td>Web Real-Time Communication</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XMPP</td>
<td>Extensible Messaging and Presence Protocol</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Live streaming\(^1\) has faced a huge growth of popularity over the recent years. This was mostly due to widely used social media services, such as Facebook, Snapchat and Twitter, incorporating live streaming tools into their platforms, providing an easy way for their users to "go live" anytime and anywhere [1] [2].

Due to the ease of "going live" whenever and wherever the users want, live streaming has become a popular trend among the attendees of large-scale events, such as music festivals or sports events, allowing them to easily share their own experiences of the event with people around the world [3]. Passive bystanders in the presence of newsworthy incidents such as natural disasters and attacks have started to make use of live streaming platforms as well, facilitating the dissemination of information regarding those incidents.

These phenomenona are part of what is called citizen journalism, which refers to amateur reporters actively participating in the process of collecting, reporting, analyzing and disseminating news and information, usually making use of new media technologies such as social media platforms. Due to the wide availability of the referred technologies nowadays, citizens are more often than not able to report breaking news more quickly than traditional media reporters. This opens the doors to new ways for making journalism, in which citizens are no longer only on the receiving end of the media system, but are also actively participating in the production of content that will be consumed by other people.

Realizing that this might be an opportunity for content providers to enrich their coverage of large events or newsworthy incidents by making use of the citizens’ contributions, MOG Technologies started a project in a partnership with INESC TEC, Jornal de Notícias and OSTV. This project, initially described as "Cloud-based collaborative content production for live events", aims to provide a set of tools that enable the collaboration between journalists and citizens in the coverage of large events.

\(^1\)Live streaming - ability to broadcast live video through the Internet, either publicly or privately, as a way to reach the intended audience.
Introduction

1.1 Context

The project "Cloud-based collaborative content production for live events" is part of a larger project – the CHIC project – which was defined by a large consortium composed by 24 entities led by MOG Technologies. The Cooperative Holistic View on Internet and Content (CHIC) project is supported by Portugal 2020\(^2\) funds and aims to revolutionize the audiovisual and multimedia sectors by developing and demonstrating a wide range of new processes, products and services in these two sectors [4].

![Figure 1.1: CHIC project’s overall structure](image)

As seen in Figure 1.1, which provides an overview of the CHIC project’s overall structure, this project was structured according to three main action areas [4], to note:

- **PPS A** – “Open platforms for managing the production and distribution of digital content in the Cloud” – that intends to provide a technological solution that allows the creation of open platforms, agnostic to the type of formats and equipment, to support different types of workflows related to content production and distribution;

- **PPS B** – “Management of contents belonging to the national cultural heritage based on open systems of preservation and interaction” – that intends to present a set of strategies, methodologies and activities for the construction of an ecosystem of platforms, products, systems and services aiming at national heritage valorization;

\(^2\)Portugal 2020 - partnership agreement between Portugal and the European Commission that defines the programming principles of economic, social and territorial development policy to promote in Portugal, between 2014 and 2020.
Introduction

- **PPS C** – “Creation, production and consumption of content, focusing on service quality and experience, using immersive and ultra-high definition environments” – that intends to develop work methodologies, technological artifacts and ways of valorizing research and development results, to increase national competitiveness concerning the use of new forms of content consumption (namely, through immersive technologies and/or high resolution content).

The project "Cloud-based collaborative content production for live events" is the activity A2 represented in Figure 1.1, being part of a set of projects comprising the PPS A that was described previously. This activity intends to provide the broadcasting and live production world with a tool to achieve a massive user engagement experience in large events, and consequently to increase their potential revenue.

### 1.2 Project

The main goal of the project "Cloud-based collaborative content production for live events" is to design and develop a cloud-based framework that allows real-time editing of multiple video streams captured in a large event by the audience using their smartphones. Henceforth, this project will be referred to as MOGPlay which was the name given by MOG Technologies to facilitate references to the activity A2. The general workflow for the MOGPlay project can be seen in Figure 1.2.

![Figure 1.2: General workflow for the MOGPlay project](image)

During the event, the users, either citizens or journalists, have access to a mobile application that connects itself to the different Wi-Fi access points available at the event, as is represented
on the left side of Figure 1.2. Using the aforementioned application, users are able to capture what they consider to be the most interesting images and stream them out to the cloud platform. The different access points define a geographical area that can be used by the event producer, or director in charge of covering the event, to preview the feeds that are being captured in a certain area.

Afterwards, the event producers can select one of the captured feeds according to its relevance to the audience, using a multi-viewer panel, which is the web-based GUI represented in Figure 1.2. The selected feed can then be fed into the Regie (Program Control Room), distributed to the Internet or to large on-site screens, providing the audience with their own point of view of the event.

At the time of the writing of this document, the existing version of the MOGPlay platform prototype offered functionality to enable participants at live events to capture audio and video and seamlessly stream them in near real-time to the cloud, using the mobile web app in their smartphones. At the production/server side, those received feeds could be easily inspected by the event producer using a drag-and-drop web-based GUI, who could thus quickly select at any given instant, the feed to be sent out as the main stream to the general public, either through broadcast channels or via Youtube Live.

It should be noted that the aforementioned prototype does not differentiate the feeds received by the platform, meaning that streams are not associated to events and that there is no distinction between the streams coming from citizens and journalists. There is also no data store associated to the platform.

The present dissertation has been conducted within the context of this project with the aim of expanding the functionalities of the solution developed by MOG Technologies.

1.3 Motivation and Objectives

In the context of the MOGPlay platform described previously, there are a wide array of different challenges that can be identified and that need to be addressed in order to fulfill the objectives defined for the project. Those challenges include:

- **The need for a communication mechanism.** This communication platform should enable the event producers to communicate with the streamers, so that the producers are able to give feedback and indications of what to do next to the mobile application users;

- **The need to identify the different types of users on the platform.** This distinction is necessary not only to distinguish streams coming from journalists and from citizens, but also to assign differing permissions to the platform’s users;

- **The need to monetize the platform.** A mechanism that enables the platform monetization is required so that it gives some kind of profit to the content providers that will use it;
Introduction

- **Filtering of web streams by event.** This filtering needs to be accomplished so that the streams collected from the event venue can be associated with the corresponding event;

- **Lack of a data store.** A data repository is needed to enable the persistent storage of the platform’s data, including information about the users, events among others.

As part of the objectives defined for the MOGPlay project, the main goal of this work is to design and develop a system to offer real-time messaging and notification capability between producers and streamers, providing a solution to the challenges mentioned previously. Thus, this work’s objectives are to:

- Provide an authentication system for the different types of users;
- Provide a real-time communication system between the event producers and streamers;
- Enable producers to create and distribute advertising to the mobile application users;
- Provide event customization options for producers, enabling stream filtering by event;
- Design a data store for the system, enabling the persistent storage of the platform’s data;
- Integrate the implemented system in the cloud platform.

There are some interesting challenges that need to be overcome and thus there is the need to conduct research in certain areas to identify possible alternative solutions to such challenges.

Firstly, one of those problems arises from the need to select the most suitable web-based communication protocols and technologies to use for the messaging system, providing low-latency in the transmission of the different types of data and near real-time communication. The communication infrastructure should also be scalable, enabling the sending of media to a lot of users without overloading the producers’ web clients. Therefore, an extensive study on web technologies and protocols that provide the aforementioned characteristics was done and an overview of this study is reported during this dissertation.

Lastly, it is important to do a research on similar platforms to understand which areas the MOGPlay platform can differentiate itself from the others, that is, to understand how this platform can provide added value when compared to other applications/marketing solutions. Therefore, a study on similar applications was done during this document.

### 1.4 Dissertation Structure

This document is organized in seven chapters, described as follows:

- **Introduction** — Chapter 1, this one, presents an introduction to the theme of this dissertation, its objectives and motivation, and the outline of the rest of the document;
Introduction

- **Real-time Web** — Chapter 2 presents a literature review of some concepts and technologies that enable real-time communication on the Web, one of the main subjects explored by this thesis;

- **Live Streaming Applications** — Chapter 3 presents an overview of similar applications, providing a point of comparison to this thesis’ work;

- **MOGPlay Platform** — Chapter 4 presents the main problems that are to be studied and dealt with during this thesis in the context of the MOGPlay project, while providing additional details about the MOGPlay platform;

- **Operate&Notify Ecosystem** — Chapter 5 presents details about the solution that was implemented in order to address the problems described in the previous chapter, while also presenting the achieved results;

- **Conclusions and Future Work** — Chapter 6 contains the conclusions that can be taken from this thesis and the future work to be done.
Chapter 2

Real-time Web

The evolution of Web applications over the recent years has turned real-time communication into a mandatory requirement to the success of many newly developed applications. Social media and instant messaging web applications are just a few examples of web-based software that are part of the day-to-day lives of millions of people around the world nowadays and that require immediate response to their users [5].

Taking this demand into account, many technologies and protocols that allow for real-time interaction on the Web have been presented over the years [6]. One of the most relevant technologies to show up recently in this context was Web Real-Time Communication (WebRTC) which is a new industry standard that enables communications of audio and/or video in real-time, without the need to install proprietary plug-ins [7]. This technology was a ground-breaking addition to the Web, enabling the implementation of peer-to-peer applications for voice calling, video chat and peer-to-peer file sharing using native browser APIs. Taking into consideration the need of a technology that enables real-time transmission of video and audio, WebRTC was studied extensively in order to fill the project’s requirements.

However, WebRTC on its own is not sufficient to build a distributed application, requiring one or more servers to deal with real-world constraints (as it will be explained in more detail afterwards). Given this, we need to ensure that the client/server communication framework is aligned with the messaging system requirements. As such, in order to find out the most suitable solution in the context of this work, an overview of different technologies that enable real-time communication between clients and servers (also referred to as push technologies) was made.

Nonetheless, some basic concepts and definitions related to the Web need to be introduced before touching the topics described above.

---

1Peer-to-peer - network architecture in which each endpoint cooperates with each other to provide services to all endpoints in the network, without a central server. Every endpoint functions as a server and as a client.
Real-time Web

2.1 Web Basics

The World Wide Web, also referred to as Web, is a network of information resources widely used to share information around the world and that is accessible over the Internet. The Web is built around three core concepts [8] [9]:

- **Uniform Resource Identifier (URI)**, a uniform naming scheme used to identify resources over the Internet;
- **Protocols** such as HTTP, used to find resources on the Web, being responsible for the communication between them;
- **Hypertext** such as HTML, that provides a representation of web pages.

In order to better understand how the Web works, it is important to provide an overview of the technologies in which the Web is mainly based, as seen in Figure 2.1.

![Figure 2.1: Web technologies [10]](image)

These technologies will be briefly explained below so as to provide more context to this work, which focus on web protocols and technologies.

**Hypertext Markup Language (HTML)** defines the basic structure of a web page using markup\(^2\). **Cascading Style Sheets (CSS)** can be used to style HTML elements on a web page, allowing color and size changes among other characteristics. As for **ECMAScript**, it enables the definition of scripts that provide dynamic content on a web page, being **JavaScript** the most well-known implementation of ECMAScript [10]. All of these technologies allow the definition of the presentation and client-side logic of web pages, as seen on Figure 2.1.

**Extensible Markup Language (XML)** and **JavaScript Object Notation (JSON)** are textual data formats that can be used for language independent messaging and to represent arbitrary data structures, being both widely used to receive data from web servers. As for **Document Object Model (DOM)**, it is a platform-independent programming interface that provides a structured representation of a document and that defines a means by which the document can be accessed by scripts, allowing them to dynamically change their content, structure and style [10].

---

\(^2\)Markup - annotation system in which tags are associated with elements of a document defining their relation to the rest of the document and how each of those elements should be presented.
Real-time Web

technologies allow the definition of data structures that can be used to receive data from servers or
to dynamically change the content of web pages.

**Uniform Resource Identifier (URI)** provides a simple means of uniquely identifying web
resources over the Internet, such as HTML documents, images among others. URIs typically
consist of three parts: the naming scheme of the mechanism used to access the resource (such as
the HTTP or HTTPS schemes), the name of the machine hosting the resource and the name of the
resource itself, given as a path. The most commonly used form of URI is the Uniform Resource
Locator (URL), which refers to a network address in which a web resource can be found over the
Internet, such as a web page [9].

**Hypertext Transfer Protocol (HTTP)** is an application-level protocol for the transmission
of hypermedia documents, such as HTML, being the foundation of data communication for the
Web. As such, this protocol was projected to transfer information between web browsers and
web servers, though it can be used for other purposes. HTTP follows the traditional client-server
model, in which the client sends an HTTP request to the server to get a resource and the server
sends the response to that request as seen in Figure 2.2 [8].

![Figure 2.2: Traditional HTTP client-server model](image)

In summary, web pages are documents formatted with HTML that allow displaying all kinds
of multimedia and software and that are identified by a specific URL. A user is able to request
a specific web page by introducing its URL in the web browser. If the web page URL is valid
and there is access to the Internet in the device in which the request was made, the web page is
rendered in the user’s web browser after processing the request. In order to display a web page to
the user, the browser needs to request it via HTTP to a web server that sends it back in response if
the request was successful, also through HTTP.

---

3 Hypermedia - medium which comprises various types of media which include text, audio, video, animations and
graphics and that enables non-linear interaction with these data types.
2.2 WebRTC

Before the emergence of WebRTC, the landscape of communications between web clients was dominated by a set of proprietary protocols and plug-ins [11]. One of the most widely-used plug-ins to this purpose used to be *Adobe Flash*, which is not supported on multiple devices, like Android or iOS, two of the most widely-used platforms [12].

This lack of support was a huge hindrance to the development of real-time interaction on the web, especially for sites that wanted to support multiple browsers and systems. Considering that most people use either their smartphones or tablets to browse the Internet nowadays, this restriction is a critical issue when it comes to the usage of *Adobe Flash* in the development of websites [11]. It was also one of the main reasons that led to the deprecation of this plug-in [12].

Taking these hindrances into account, a technology was created in an attempt to simplify the development and deployment of real-time communications using built-in functionalities of web browsers, without the need to install third-party software or plug-ins. This technology was called WebRTC which is an open-source technology that enables real-time audio, video and data communication between web clients via simple *JavaScript* APIs [7].

WebRTC was created by Google and is being standardized by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF). W3C is responsible for the specification of the browser APIs while the IETF Working Group is working on defining the specifications regarding the protocols and session establishment in WebRTC [13].

WebRTC is currently facing an exponential growth in its adoption [14], despite its still existent browser compatibility issues (particularly with older browsers). However, a huge effort is being made into overcoming this issue and it is predicted that this will cease to be a problem some years from now, as recent versions of all major browsers (*Google Chrome*, *Firefox*, *Opera*, *Edge* and *Safari*) already support WebRTC [15].

Another evidence of WebRTC’s growth is its usage in widely-used applications such as *Google Hangouts*, *Facebook Messenger*, *Appear.in* and *Discord* [16] [17].
2.2.1 General Architecture

The architecture of WebRTC consists of an extensive variety of different standards, covering both the application and browser APIs. The overall architecture of WebRTC can be seen in Figure 2.3.

![WebRTC architecture](image)

The voice and video engines are frameworks responsible for transmitting audio and video streams from the sound card and camera to the network. As for the transport components, they allow establishing connections across various types of networks.

The voice and video engines in WebRTC were designed in order to keep latency as low as possible, while still providing the best conceivable streaming quality for both video and audio. To this end, both of these engines make use of codecs\(^4\) that enable bandwidth-adaptive bitrate, that is, that adjust the transmission bitrate to the always changing bandwidth and latency between the clients. The audio and video engines also process the audio and video streams to increase their quality and synchronize them [18] [19].

The voice engine in WebRTC is able to increase audio quality by applying codecs that hide the negative results of network jitter and packet loss and that also remove echo and background noise in the audio stream. As for the video engine in WebRTC, it provides a set of software that help conceal the effects of jitter and packet loss and that removes video noise from the image capture by the webcam [18].

\(^4\)Codec - software that is able to compress large amounts of data for transmission, such as audio and video, and to decompress back received data in order to play it in its original format.
Real-time Web

In order to transport the information from and to the participants in an established connection, WebRTC makes use of a set of protocols, which are:

- **Real-time Transport Protocol (RTP)**, used with the objective to send the data from and to the WebRTC clients in near real-time;
- **Real-time Control Protocol (RTCP)**, used to provide information about the transmission quality;
- **Secure Real-time Transport Protocol (SRTP)**, used to keep the safety of that same data, providing encryption, message authentication and integrity and replay attack protection;
- **STUN, TURN and ICE protocols**, used to create peer-to-peer connections as it will be explained further below in this document.

### 2.2.2 WebRTC Application

Although WebRTC enables web clients to communicate directly with each other in a peer-to-peer fashion, a WebRTC application still needs intermediate servers to coordinate communication between the web clients, a mechanism commonly referred to as signaling [20], and to deal with NATs\(^5\) and firewalls\(^6\) [21]. The basic architecture for a WebRTC application involves at least three parties: a signaling server and two peer web clients, as seen in Figure 2.4. Note that this architecture does not take into account the issue of traversing NATs and firewalls for now.

\(^5\)Network Address Translation - method that consists of translating a computer’s IP address in a local network (internal address) to a single IP address in a public network which can be accessed by other devices on the Internet.

\(^6\)Firewall - network security device that blocks the access of unwanted content, without hindering the network traffic of content that can be trusted. It is widely employed in most devices with access to the Internet.
In order for a WebRTC application to set up a data path between the web clients, they need to exchange information with each other, such as the session control messages to open and close connections, error messages, media metadata (mainly information about codecs), security data (to establish secure connections) and network data (network information such as the host’s IP address and port). The referred data should be encapsulated using the Session Description Protocol (SDP) format [21], which is a format that describes the parameters necessary to establish media communication sessions between two or more endpoints. As to how this exchange of information functions in a WebRTC application, firstly a participant extends an offer with the aforementioned data to the other peers and receives answers in response to their offer from the other participants as can be seen in Figure 2.5.

While the direct communication between web clients can be achieved using only WebRTC, this technology does not define how participants should exchange data between each other and how to find peers that want to communicate in a network, needing an intermediate server (a signaling server) to deal with these issues. The technologies to use for the signaling server are not defined by WebRTC and it is up to the developer to choose the most appropriate ones to use taking into account the requirements of the WebRTC application to implement [22].

It is also important to ensure that the signaling mechanism is secure. As such, the signaling server should use secure protocols [21], such as Hyper Text Transfer Protocol Secure (HTTPS), which is an extension of the HTTP protocol. HTTPS adds a layer of security to HTTP by encrypting the messages exchanged between the browser and the web server, ensuring that messages cannot be intercepted before the encryption.
Core APIs

In order to better understand how WebRTC works, it is important to mention and briefly describe the core API calls of WebRTC, which are:

- **GetUserMedia** — allows retrieving video and audio streams (also called media stream tracks) from the camera or microphone hardware of the user’s device [11];

- **RTCPeerConnection** — represents a peer-to-peer communication. It does a lot of work unbeknownst to the developer such as packet loss concealment, codecs’ execution, bandwidth administration, encryption, etc. [23];

- **RTCDataChannel** — allows sharing of any type of data between already connected peers.

The data channel, RTCDataChannel, is created as a generic transport service using the Stream Control Transmission Protocol (SCTP), allowing configurable delivery semantics, providing features from both UDP 7 and TCP 8. To ensure confidentiality and authentication of SCTP packets, they are sent using a DTLS (Datagram Transport Layer Security) protected association [11].

General Implementation

In order to set up a connection between WebRTC peers and exchange data between them, three general tasks are to be accomplished:

- **Create a RTCPeerConnection for each end of the call.** At each end, either add the local audio/video stream from GetUserMedia or create a data channel, using RTCDataChannel, on top of it in order to transmit arbitrary data;

- **Get and share network information.** The potential routing paths should be identified for the peer candidates on both sides of the connection and this information should be relayed to all the peers. The Interactive Connectivity Establishment (ICE) protocol can be used in order to discover WebRTC peers on the network;

- **Get and share local and remote descriptions.** Both ends of the connection need to agree on the data to be transmitted/received before establishing the connection and this is accomplished by negotiating the connection parameters using the SDP format.

---

7 User Datagram Protocol - simple communication protocol which works over the transport layer and that does not guarantee the delivery of all packets nor their order. Widely used in video conferencing and voice over IP applications.

8 Transmission Control Protocol - communication protocol which works over the transport layer and that provides reliable, ordered packet delivery and error detection and correction. Its uses include the Web and e-mail applications.
Traversing NATs and Firewalls

A WebRTC application can use the ICE framework in order to traverse NATs and firewalls, by providing one or more server URLs to the application corresponding to the STUN and/or TURN servers that will be able to properly relay data from and to the peers.

To discover possible endpoint candidates, initially ICE inquires the operating system for local IP addresses and tries to make a connection using them. If that fails, which it will for devices behind NATs since their private addresses cannot be used directly, then ICE tries to obtain a public address using an external STUN server. If that also fails, traffic is routed via a TURN relay server, but only as a last resort as this server is very resource intensive [21].

2.2.3 Signaling Protocols

As it was mentioned before, even if WebRTC solves a lot of problems related to the peer-to-peer multimedia transmission, the network-related aspects are mostly out of the scope of this technology [22]. As such, the choice of the most suitable protocol to implement the signaling mechanism in WebRTC is a subject that requires further study.

Signaling is needed to negotiate session parameters between two or more web clients. This mechanism should allow the identification of other peers within a connection, to figure out whether the other party wishes to communicate and to agree on technical parameters for the communication.

As for the implementation of the signaling mechanism in a WebRTC application, it is up to the developer to choose the signaling protocol or alternatively implement a custom signaling mechanism tailored to the application [22]. SIP and Jingle (a XMPP extension), two of the most popular signaling protocols, will be briefly presented below.

SIP

Session Initiation Protocol (SIP) is an application-level signaling protocol widely used for Internet telephony and videoconferencing systems [19] [21] and allows setting up, modifying and terminating multimedia sessions. It can be used for an extensive variety of applications such as voice, video and messaging applications.

The protocol is similar to HTTP in the sense that it uses a request-response pattern and features status codes. Furthermore, SIP is not linked to any particular transport protocol, so it can used over HTTP or WebSockets [13].

Jingle

Jingle is a signaling extension for the XMPP protocol, used for session management of voice, video, file transfer, and other applications [18] [24]. As such, Jingle can be employed to initiate
and control media sessions between two peers that are XMPP entities in a way that is interoperable with prevailing Internet standards [13]. It is compatible with SIP and it can be used over a variety of transport protocols, such as HTTP and WebSockets.

As for Extensible Messaging and Presence Protocol (XMPP), it is an open source, extensible protocol for streaming XML elements in real-time between any network endpoints [25]. XMPP is mainly used for presence (the users’ online status) and instant messaging, even though XMPP supplies a generalized framework to exchange XML data. Some widely-used applications that use XMPP include WhatsApp and Nimbuzz.

2.2.4 Media Server

While the peer-to-peer architecture for a WebRTC application shown in Figure 2.4 is enough for use cases which involve only one-to-one communication between web clients, the same is not true for use cases involving WebRTC broadcast in an one-to-many fashion. While a broadcast would not be a problem when sending text messages, the same does not apply when sending audio/video data as this transmission would require larger bandwidth as the number of recipients grows since the peer-to-peer architecture is not scalable [26].

That is, let us assume that one of the requirements of a WebRTC application is to broadcast a low-resolution video. For each transmission of the video from the broadcaster to a single recipient, a network capacity of 500 kb/s (kilobits per second) would be required in the broadcaster’s web client so as to obtain good results for the recipient.

Assuming that there are 100 recipients for a broadcaster’s stream, the needed network capacity for the sender’s web client would have to be around 50 Mb/s (megabits per second) just to send the video, as it is represented in Figure 2.6. And as the number of broadcast recipients grows, the more network capacity would be needed just so that the broadcaster is able to broadcast the video in low resolution [26]. If we need a higher quality stream, then this scenario would only get worse as the required network capacity would increase drastically.

Figure 2.6: WebRTC broadcast with no server, adapted from [26]
Although the adoption of fiber-optic communication\(^9\) has been growing over the recent years, providing high-speed Internet connections, the number of possible recipients be highly limited if the scalability of the communication infrastructure was not taken into account when developing the WebRTC application. This would also have a negative effect on the streaming quality in the web clients receiving the media in real-time as it would be lower than intended. This scenario becomes significantly worse when the broadcasting is made using a 3G network [26].

Taking all that has been said into account, it can be concluded that for use cases involving media broadcast, it is essential to provide the application with an intermediate server that takes care of media stream processing and routing in real time, which can be referred to as a media server. The existence of a media server or a number of them would be enough to deal with the aforementioned scalability issue, providing a way to decide how much bandwidth each recipient will consume, the machine type of the server among other issues [26].

There are two main different approaches to routing in media servers, which makes it possible to divide media servers into two major categories: Selective Forwarding Unit (SFU) and Multipoint Control Unit (MCU). A SFU media server is able to receive various media streams and then decide which of the referred media streams should be relayed to the peers. On the other hand, in a MCU approach, every participant sends their media streams to a central media server (the so-called MCU) which sends back a single stream to each participant after mixing all of the streams it received into a single one [27].

While MCU provides the best solution when it comes to network load, it is also very expensive to operate since the MCU server needs to decode, compose and encode again the media it receives for all peers. As such, SFU is usually considered the most cost-effective solution to media servers as it is not only much more scalable than the peer-to-peer architecture but is also relatively low-cost [27].

Currently, there is a wide variety of open source solutions for media servers which include the Jitsi Platform, Kurento Media Server (also known as OpenVidu), Janus WebRTC gateway (also known only as Janus), Medooze and mediasoup. In order to choose the most suitable media server out of all the available solutions, there are a wide variety of characteristics to take into consideration, the most important and general ones including the popularity of each open source solution, whether their code-base is up-to-date and how good their documentation is (if it exists). It is also important to check how well do the open source media servers scale as the number of participants increase [28]. In order to determine and compare the scalability of the referred solutions, a study was conducted as seen in Figure 2.7.

In the referred figure, the results from various tests performed using the media servers mentioned previously can be seen in terms of bits per second (bps). As it can be deduced from the figure, as the number of participants increase, each connection will tend to be served with a lower bit rate. However there were some media servers which had better results, that is, higher bit rates.

---

\(^9\)Fiber-optic Communication - mechanism of data transmission over fiber by turning electronic signals into light. Provides ultra-high speed over large distances.
Real-time Web

even with lots of participants. That was the case for Janus and mediasoup which had the best results out all the tested libraries. However, considering that Janus is more widely-used (being even used in well-known commercial solutions like Slack), Janus is the solution that currently provides one of the sturdiest solutions to media servers.

![Bit rate comparison for WebRTC media servers](image)

Figure 2.7: Bit rate comparison for WebRTC media servers [29]

It should be noted, however, that Janus by itself is not only a media server, serving a whole lot of other purposes besides that as it is a general purpose WebRTC server. For example, one of its main uses is to simplify the implementation of the signaling mechanism needed in order to communicate between WebRTC clients. Janus also provides a set of server-side plugins that can be called by the web clients using the Janus’ JavaScript library to implement a wide range of media applications, such as video conferencing, voice calls, among many others.

2.2.5 Possible Alternatives

The usage of WebRTC is ideal for applications that aim to provide voice or video calls, peer-to-peer messaging or file sharing [30]. When it comes to web-based live streaming however, there are another alternatives that are commonly used besides WebRTC. As this technology focuses on transmission speed over stream quality, the usage of other technologies besides WebRTC for live streaming is still common practice nowadays. The most popular alternatives in this context include RTMP, HLS or MPEG-DASH.

The three referred alternatives will be explained briefly below:

- **Real-Time Messaging Protocol (RTMP)** – designed for low-latency transmission of audio, video and data over the Internet between a Flash player and a server. While it was initially launched as a proprietary protocol, nowadays it is available as an open specification to create
products that focus on video delivery [31]. However, due to the deprecation of Flash, CDN support for RTMP streams has been declining over the recent years [32];

- **HTTP Live Streaming (HLS)** – HTTP-based protocol used to stream live video and audio over the internet. It provides a reliable, cost-effective means of delivering content via standard HTTP web servers. Besides that, this protocol also allows a receiver to adjust the bit-rate of the media to the current network conditions, providing the best possible quality (adaptive bitrate streaming) [33];

- **MPEG Dynamic Adaptive Streaming over HTTP (MPEG-DASH)** – open-source standard used to deliver multimedia over the Internet. In MPEG-DASH, the client device plays a fundamental role by providing the intelligence that leads the video adaptation [31]. It also supports adaptive bitrate streaming, HTTP delivery and a number of other features [34].

Despite these protocols being viable alternatives when it comes to live streaming, these technologies are not suitable choices in the context of applications requiring bidirectional, real-time communication. This is because the alternatives mentioned previously only allow the transmission of audio and video from the broadcaster to the viewers and not the simultaneous transmission of media from the other side, that is, from the viewers to the broadcaster.

Another pivotal point to consider when comparing WebRTC with its alternatives is that this technology enables low-latency transmission of both audio and video, which is something that is much harder to achieve with HLS or MPEG-DASH which are usually associated with high latencies [32].

### 2.3 Client-Server Real-Time Web Technologies

The choice of the most suitable high-level transport protocol to use for the client-server communications is another important subject of study in order to get good results in the implementation of the communication infrastructure.

The message exchange provided by the transport protocol must be bidirectional, that is, it must allow the client to send messages to the server and the server to send messages to the client [21]. While client-to-server updates are easily accomplished with the traditional HTTP client-server model by sending client requests to the server requesting a response, the opposite is much harder to achieve as HTTP does not provide a way for the server to instigate a new connection to the client [19]. To overcome this drawback, Server Push Technologies have been proposed over the years, which allows emulating real-time and bidirectional message exchange [35]. As such, an extensive study on Server Push Technologies is hereafter presented in this section.

---

10Content Delivery Network - Geographically distributed network of proxy servers and their data centers, which provide a shared distribution infrastructure in multiple locations close to end-users.
Real-time Web

One of the libraries that makes use of some of the server push technologies mentioned previously to establish a bidirectional connection is **Socket.IO**. Therefore, this library is briefly presented below.

### 2.3.1 Server Push Technologies

The traditional HTTP client-server model in which messages are processed synchronously is not sufficient for real-time applications that require immediate response to the users’ actions. This is where server push technologies come in, allowing the delivery of messages from the server to the client [35]. This enables bidirectional and real-time communications in the Web, allowing a client to receive updates from the server without having to request them.

As such, a review of server push technologies with these characteristics is presented in this section. The protocols that are to be discussed in this section are some of the most used which include HTTP Polling, HTTP Long-Polling, Server-Sent Events and WebSocket Protocol.

**HTTP Polling**

HTTP Polling is an AJAX-based\(^\text{11}\) technique in which the client will emit a HTTP request to the server periodically, asking whether there is new data every few seconds as is illustrated in **Figure 2.8**. If there is no new content available, the server will send an empty response; otherwise, if the requested content is available, the server will send the the requested data [36].

![Figure 2.8: HTTP Polling in a web application [37]](image)

---

\(^{11}\)Asynchronous JavaScript And XML - methodological use of technologies provided by browsers, such as JavaScript and XML, to make web pages more interactive with the user, using asynchronous requests.
Real-time Web

If updates to server resources occur during the polling interval (the waiting time between two requests), then there will be a delay until the client receives the requested update from the server. This is obviously a big problem since the immediate responsiveness required by real-time applications will not be guaranteed. This can be solved by reducing the polling interval, but this will make the mechanic too resource intensive and inefficient, since the client requests will keep hitting the server and getting empty responses when there is not any new data. As such, while this approach is very simple to implement it is not often the most suitable choice for real-time applications, unless there is a high tolerance for message latency in them [37].

HTTP Long-Polling

HTTP Long-Polling is a technique in which the client requests data from the server similarly to HTTP Polling, but the server does not send the response to the client until the requested resource is available or a connection timeout is reached [35].

After sending the request, the client has an open connection to the server and is able to accept new data once the server is ready to send it. If the timeout is reached in the meantime however, the server sends an empty message instead of the requested resource. After receiving the response from the server, whether empty or not, the client sends another request right after, as seen in Figure 2.9.

![Figure 2.9: HTTP Long-Polling in a web application [37]](image)

In this set up, this approach allows minimizing the usage of resources and the message latency when compared to HTTP Polling in most use cases. As such, Long-Polling is a major improvement to the traditional Polling technique especially when it comes to real-time communications since the server updates are sent as soon as they are available [35]. Despite this, Long-Polling does not
provide an optimal solution as the client would still have to send a request to the server to get updates. Furthermore, if the updates to the request resources are too frequent, then the number of ensuing client HTTP requests would probably be higher than if the traditional HTTP Polling technique was employed [37].

Server-Sent Events

W3C defines Server-Sent Events as a browser API that enables servers to push notifications to Web pages without previous requests from the clients. That is, Server-Sent Events allow a web page to get updates from a server asynchronously, using only a single unidirectional channel as seen in Figure 2.10. This differs from the polling techniques mentioned previously, since the client does not need to ask the server if there are any updates available in order to retrieve them [35] [38].

![Figure 2.10: Server-Sent Events in a web application [39]](image)

However, not all browsers support Server-Sent Events, such as Edge and Internet Explorer. In these browsers a different approach should be applied to emulate this technique named polyfill, that consists in implementing a feature on web browsers that not support it.

Considering that the client-server communication needs to be bidirectional, Server-Sent Events must be used in combination with AJAX in order to build a service to exchange messages in both directions. This implementation should be implemented in a way that allows AJAX to be used to send messages from the client to server while Server-Sent Events would be used to receive updates from the server to the client in real-time [21].
WebSocket Protocol

The WebSocket protocol, standardized by HTML5, allows the establishment of a full-duplex, two-way connection with a remote web server using a single TCP socket. That is, the WebSocket protocol opens a session from a client to the server, leaving it open until the session is closed [6] [40]. After establishing the connection, the server is able to send updates to the client, without the need of extra resources [35] as seen in Figure 2.11. Text and binary data can be sent over this connection.

Figure 2.11: WebSocket protocol in a web application [39]

A WebSocket connection is initiated by the HTTP handshake procedure, during which the client sends to the server an HTTP Upgrade header to switch from HTTP to the WebSocket protocol. After receiving the request, the server sends a HTTP response to the client. Even though the WebSocket protocol differs from the HTTP protocol, WebSocket can be created over standard HTTP or HTTPS ports (80 or 443, respectively) and, as such, is able to deal with firewalls, NATs or HTTP proxies [6].

The WebSocket protocol also provides reduction in latency due to there being less message overhead [6] [41]. That is, after establishing a connection with the handshake mechanism, the messages exchanged in the WebSocket protocol only have the minimal information necessary, while HTTP messages usually carry lots of additional, unnecessary header data [42].

Lastly, it is important to remark that all browsers that support WebRTC also support WebSocket, both on desktop and mobile. The Transport Layer Security (TLS) protocol should be used along with the WebSocket protocol in order to ensure that messages cannot be intercepted while not encrypted and also to reduce problems with proxy and NAT traversals [21].
2.3.2 Overview of Server Push Technologies

Many studies over the years such as [42] and [43] have compared WebSockets to HTTP Polling and Long-Polling for usage in real-time applications. An overview of these studies will be presented below.

As noted by the authors of [42], polling techniques generate a lot of unnecessary header data while sending HTTP messages back and forth, generating up to 665 megabits per second of header data in the use case in which 100,000 clients poll to the server every second asking for updates. In comparison to the same use case, the network throughput for WebSockets is only 1.526 megabits per second, being a major improvement in terms of network traffic. A graphic providing a comparison for three use cases in which clients receive server updates every second can be seen in Figure 2.12, being that the number of clients receiving updates per second varies for each use case: 1,000 clients in use case A, 10,000 clients in use case B and 100,000 clients in use case C. Note that Polling refers to HTTP Polling in these use cases.

As such, taking into account the results of their study, the authors of [42] state that WebSocket provides a lot more scalability than polling techniques, due to the reduced network traffic verified even when lots of clients are connected to a server. In this study, the authors of [42] also compared the latency of Polling and WebSocket applications as seen in Figure 2.13 and they argue that polling introduces extra latency due to the necessity of sending a request to the server before getting the response, while in WebSockets the server-to-client messages are sent directly after the connection is established.
However, Pimentel and Nickerson noticed in [43] that the work presented in [42] does not study the effects of latency in wide area networks when using WebSockets. As such, the authors of [43] conducted a study in order to evaluate the latency in long-distance communications using HTTP Polling, HTTP Long-Polling and WebSockets, developing an application with three different versions using the three protocols referred previously.

In order to estimate the latency in all versions, the authors of [43] first had to synchronize the computer clocks over the Internet, using the Network Time Protocol (NTP). As for the results, the conducted tests revealed that the average latency associated with the HTTP polling interval is significantly higher than the one associated with either HTTP Long-Polling or WebSocket. It also showed that WebSocket had similar results to HTTP Long-Polling, except for longer distances in which WebSocket performed significantly better. As such, the authors of [43] concluded that, on average, the WebSocket protocol provides lower latency when compared to the polling techniques.

Some studies have also included Server-Sent Events in the overview of server push technologies such as [35]. The authors of [35] provide an overview of server push technologies indicating their strengths and weakness. This overview includes HTTP Polling, HTTP Long-Polling, Server-Sent Events and WebSockets besides other less known techniques which are not addressed in this work. The Table 2.1 provides an overview of the strengths (represented by △) and the weaknesses (represented by ▽) of the server push technologies presented in this work.

In order to better understand the Table 2.1, it is necessary to understand the meaning of the criteria that were selected by the authors (except the unidirectional criteria, which was added by this thesis’ author):

- **Hand-Shake** — evidences whether a client needs to establish a connection with the server through a hand-shake request;
Real-time Web

- **Persistent Connection** — indicates whether a server keeps a connection open until the requested data is available;

- **Polling-based** — indicates whether the client has to send requests to the server periodically in order to fetch new data;

- **Messages Exchanged** — indicates whether the number of messages exchanged between the client and server are low, medium or high;

- **Unidirectional** — indicates whether the communication established is unidirectional or not (if it is not, then it is bidirectional).

<table>
<thead>
<tr>
<th>Name</th>
<th>Hand-Shake</th>
<th>Persistent Connection</th>
<th>Polling-based</th>
<th>Messages Exchanged</th>
<th>Unidirectional</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP Polling</td>
<td>△ No</td>
<td>▽ No</td>
<td>▽ Yes</td>
<td>▽ High</td>
<td>▽ Yes</td>
</tr>
<tr>
<td>HTTP Long-Polling</td>
<td>△ No</td>
<td>△ Yes</td>
<td>▽ Yes</td>
<td>▽ Medium</td>
<td>▽ Yes</td>
</tr>
<tr>
<td>Server-Sent Events</td>
<td>▽ Yes</td>
<td>△ Yes</td>
<td>△ No</td>
<td>△ Low</td>
<td>▽ Yes</td>
</tr>
<tr>
<td>WebSockets</td>
<td>▽ Yes</td>
<td>△ Yes</td>
<td>△ No</td>
<td>△ Low</td>
<td>△ No</td>
</tr>
</tbody>
</table>

Table 2.1: Overview of server push technologies, adapted from [35]

As seen in the Table 2.1, Server-Sent Events are very similar to WebSockets when it comes to their strengths and weaknesses, the only exception being that WebSocket connections are bidirectional while Server-Sent Events are unidirectional. In fact, both Server-Sent Events and WebSockets enable high-performance communication, being both good alternatives for low-latency streaming of data.

Nonetheless, the fact that Server-Sent Events only provide a unidirectional channel makes it harder and unnatural to implement as it would have to be used along with AJAX in order to provide a bidirectional messaging mechanism [21] in real-time applications.

Therefore, taking all that has been said into account, WebSockets provide the most suitable solution to build scalable, real-time applications, all the while providing a bidirectional channel between the client and server.

2.3.3 *Socket.IO*

*Socket.IO* is a *JavaScript* library that enables real-time, bidirectional and event-based communication between the browser client and the server. This library is constituted by two parts as can be seen in Figure 2.14: a server-side library available for *Node.js*\(^\text{12}\) and a client-side library available for the web clients [44].

\(^{12}\)*Node.js* - open-source *JavaScript* runtime environment that allows for the execution of *JavaScript* code outside of a browser, focused on migrating *JavaScript* programming from the client-side to the server-side. Allows for the creation of highly scalable, event-driven applications, being widely employed due to these characteristics.*
The client-side part of Socket.IO primarily uses the WebSocket protocol to connect to the server, but in the event that WebSockets are not supported, it can fall back onto other methods, such as Adobe Flash sockets and HTTP Long-Polling. It provides a wide range of features including connection reliability, auto-reconnection support, disconnection detection, broadcasting to multiple sockets among others [44].

Considering that Socket.IO mostly uses WebSockets as its transport protocol (which was shown to be the most suitable protocol to use in real-time applications out of the server push technologies previously presented) while providing fallback to other protocols when the WebSockets protocol does not work, this is a good solution to be taken into account for real-time applications [46].

2.4 Conclusions

WebRTC is an emerging technology that currently offers the most robust solution to file sharing, voice calling and video chat applications over the Internet. This technology was considered to be ground-breaking in the sense that it allowed the implementation of these functionalities on the Web without the need to install additional third-party software or plug-ins, which was not the case before.

Nonetheless, WebRTC on its own is not enough to build a working distributed application, having a need for intermediate servers that handle signaling between web clients and traversing NATS/firewalls. In use cases involving media broadcasting, a media server is also needed to deal with scalability issues. After reviewing existing solutions to media servers, it was concluded that Janus WebRTC Gateway currently provides the sturdiest solution to this issue. This solution also simplifies the implementation of the signaling mechanism necessary for WebRTC applications.

In order to get good results in the implementation of the communication infrastructure, it was also important to review web technologies that enable real-time client-server communications. To allow this, the message exchange in the client-server architecture must be bidirectional, receiving real-time updates from the server.
However, real-time server updates are usually not allowed by the traditional HTTP client-server model, which works as seen in Figure 2.2. To solve this limitation, server push technologies have been proposed over the years, which allow the server to proactively send messages to the client, as is intended. Some of the most used server push technologies include HTTP Polling, HTTP Long-Polling, Server-Sent Events and WebSockets, which were presented throughout this section.

An overview of the aforementioned server push technologies was also made in order to determine the most suitable choice to use for this thesis’ work. It was concluded after reviewing a set of studies that WebSockets currently provide the most complete and efficient solution for real-time client-server communications.

It was also deduced that Socket.IO, a Node.js library, provides a sturdy solution for real-time communications as well, since not only does this technology mainly use the WebSockets protocol, it also provides fallback to other technologies when WebSockets are not available.
Chapter 3

Live Streaming Applications

A research was made on existing platforms or marketing solutions that are used in the context of live streaming with particular emphasis given to the messaging component present in these platforms. This study was done in order to determine what the available solutions have to offer when compared to the MOGPlay platform and to understand which of its features make this project stand out from the others.

The applications that are to be presented and discussed were divided in two major types: consumer oriented platforms and business to business (B2B) platforms. The first type of applications are mainly meant to be used by consumers for personal use while the latter type are primarily aimed at other companies, such as content providers.

Finally, a comparative analysis was done in order to provide an overview of all the applications that were discussed during this section and compare them to the MOGPlay platform.

3.1 Consumer Oriented Platforms

In this section, an overview of commercial, consumer oriented applications that are mostly used in the context of live streaming are presented and discussed. As there are a wide variety of existing platforms that offer these characteristics and that are similar to one another, only some of them are presented: Facebook Live and Twitch.

Regarding Facebook Live, this application serves a similar purpose to the MOGPlay platform, being widely used for covering large-scale events or newsworthy incidents (citizen journalism). As for Twitch, this platform underscores the importance of a messaging system in live streaming solutions as will be clarified below.

Discord is also another platform that is discussed during this section. While not used in the context of live streaming, this widely-known platform makes use of technologies that are relevant in the context of this work, besides providing a messaging system which is one of the central themes of this work.
3.1.1 Facebook Live

Facebook Live is the video streaming service of Facebook which was made public in January 2016. It allows its users to broadcast directly from the Facebook application on the user’s mobile device, either Android or iOS. In March 2017, live streaming support was extended to the web, allowing the user to stream directly from the Facebook website.

The streams appear on Facebook’s news feed and other users can comment and react to them while the streamer is filming. The comments and reactions that are illustrated in Figure 3.1 are displayed to the broadcaster in real-time while they are making the transmission. After the broadcast ends, the video is published to the streamer’s page so that fans or friends who missed it can watch later. Also, the broadcaster can remove the video post at any time, just like any other post on their page.

![Facebook Live’s user interface](image.png)

Figure 3.1: Facebook Live’s user interface [47]

The mobile application of Facebook Live is widely used by a lot of its users to stream large-scale events or newsworthy incidents in real-time, being one of the most used tools in the context of citizen journalism [48]. In fact, there have been many cases in which live streams from Facebook were displayed when disseminating a news item (e.g., transmissions containing footage from a
newsworthy incident like a natural disaster) or were even the main theme of some news due to its exceptional content (e.g., transmissions containing violent content).

3.1.2 Twitch

Twitch is an online video platform focused on live streaming that was launched in 2011 and that is currently the leading social video service and community for video game culture. The website is mainly focused on video game live streaming, including broadcasts of eSports\(^1\) competitions and video game playthroughs, though it has been used for other purposes including music broadcasts.

The possibility of receiving feedback was one of the features that enabled the growth of Twitch’s user base [49]. Thus, Twitch provides an interesting example about how the interactivity of live chat can motivate users to create live transmissions.

As a whole, Twitch provides the following features:

- Enables registered users to broadcast live video;
- Enables streamers to archive videos that were previously broadcast;
- Allows video transmissions to be viewed live or via video on demand\(^2\);
- Enables users to follow and message each other;
- Enables users to participate in chat rooms available in each live stream;
- Allows users to support their favorite channels by making donations or by purchasing Bits (virtual currency that can be donated to streamers).

3.1.3 Discord

Discord is a VoIP\(^3\) application and digital distribution platform popular with gamers and streamers that was launched in 2015. The application is supported on a wide variety of platforms, being available as a Web app and as a native app on desktop and mobile [16].

The Discord platform specializes in text, image, video and audio communication in a chat channel, being usually used by online gamer groups so that they can communicate with each other during games [50]. Considering that the application is to be used by gamers that usually play a game as a team, low-latency data transmission is a key feature of this application. Low-latency communications are fundamental so that the players can coordinate their tactics in real-time (usually in voice calls).

---

1 eSports - organized competitions of electronic games between professional players.  
2 Video on demand - system in which users are able to select and watch archived videos whenever they want.  
3 Voice over IP - methodology that allows voice communication over the Internet.
The Discord’s web application uses WebRTC to implement the voice call and video chat functionalities. Thus, WebRTC is responsible for the delivery of real-time audio and video communication to the users. Besides that, the connections from the web clients to the Discord servers are all done using the WebSocket protocol. The communication signaling implemented in the Discord platform is custom-made as well in order to satisfy the application’s requirements [16].

Taking what has been said into account, the Discord’s web application provides an interesting use case for the usage of WebRTC (for the multimedia transmission) alongside the WebSocket protocol (for the client-server communications), enabling low-latency transmission of different types of data when used together. Bearing in mind the success and wide adoption of this platform, Discord provides a significant example of a successful commercial application that makes use of this architecture.

### 3.2 Business to Business (B2B) Platforms

In this section, a brief overview of marketing solutions that are mainly used in the context of live streaming and that are targeted to other companies (business to business platforms) are presented and analyzed.

The platforms which are to be discussed during this section include Livestream, Bambuser and DaCast. The Livestream and Bambuser solutions provides most of the features provided by the MOGPlay platform, being very similar to it. As such, it was important to present these two solutions in detail. As for DaCast, this solution offers compelling examples about how to achieve monetization on a live streaming platform.

#### 3.2.1 Livestream

Livestream is a video live streaming platform launched in 2007 that allows their customers to broadcast live video content through the Internet and the viewers to play the content via a wide variety of platforms, including the Web, iOS and Android.

Livestream provides a completely integrated end-to-end live video solution for businesses of all sizes. They supply live production video switchers4, video and audio mixers, wireless camera streaming accessories and a cloud-based broadcasting and video content management platform. They also provide professional services and, for those customers interested in a complete solution, a full-service professional production company.

Livestream offers different software products depending on the user’s profile and target device, to note:

- **Producer Desktop Software** – free desktop application meant for entry-level costumers available on Windows and Mac. This platform is able to receive live video and audio from

---

4 Video switcher - device that receives multiple incoming video signals from various sources and directs one of those signals to a single output (e.g., a monitor device).
webcams or professional cameras and relay it to the Livestream cloud-based broadcasting platform. This application also allows users to broadcast screen recordings;

- **Producer Mobile Application** – free mobile application available on Android and iOS. This application enables their users to broadcast live video, chat with their audience while live, share photos and text updates and see their own and other users’ live streams;

- **Livestream Studio** – the company’s professional live production software package, allowing producers to mix and edit live video streams and deliver the video output to the Livestream cloud or other platforms including Facebook Live and Periscope. Live video feeds can come from all types of cameras whether wired or remote, that is, the live stream can come from cameras of remote devices like a mobile phone. Thus, the software supports frame-accurate high-definition live switching (as seen in Figure 3.2). This product requires a paid subscription.

![Figure 3.2: Livestream Studio’s user interface](image)

The Livestream cloud-based service is an online platform used by broadcasters to manage and deliver live video. The platform has a variety of features and functionalities including video hosting, video embedding on any website and the delivery of live streams to multiple platforms. Besides that, it also provides their costumers the ability to monetize their videos either via advertising or via paywall\(^5\) integration. The Livestream cloud also enables its users to create events,

\(^5\)Paywall - feature that enables the monetization of content on websites by requiring payment before the users can access it or use it.
associating it with a start and end date. The broadcasters can then broadcast to one of the created events using either the Producer Desktop Software or the Producer Mobile Application. The Livestream cloud service also requires a paid subscription which is more expensive the more features it provides.

The main features of the Livestream platform as a whole are the following:

- Enables broadcasters to setup and deliver live events easily with video hosting delivered via CDNs, enabling high volume global video delivery;
- Provides broadcasters the possibility to interact with their audience;
- Enables event creation on the cloud so that broadcasters can make streams associated to the created event;
- Allows users to custom brand the video experience including video player customization and to place the video embeds across the web;
- Enables advertising integration (for example, with Google Ad Manager);
- Provides social media integration with all the popular platforms.

3.2.2 Bambuser

Bambuser is a live streaming platform launched in 2007 that enables interactive live video broadcasting from mobile phones and webcams to the internet. Bambuser’s strengths include overall ease of use, support for over 300 devices for broadcasting and playback, the ability to deliver multiple video streams and a compact embedded player.

The Bambuser platform consists of a wide range of products, namely:

- The Broadcaster App, a mobile application meant for recording and broadcasting;
- The Content Manager, an online platform serving as a centralized hub for all incoming streams and content;
- A advanced set of developer tools, SDKs and APIs, designed to integrate real-time video and photo contribution capabilities within native apps and online platforms. These developer tools can be used to aggregate user generated content from the public, which can then be published to any application or platform, including Facebook Live and other social media platforms.

The Bambuser platform is a solution to facilitate the creation, management and distribution of mobile live video and photo content, generated by anyone from members to staff, to members of the public, via their standard mobile devices. The main features of this platform are:

- Allows users to publish live and recorded videos to virtually any application or platform;
Live Streaming Applications

- Provides the location of the streamers using the mobile web application to the Content Manager web app;
- Ability to combine different filters to narrow down content on the Content Manager web app. Provides filters by author, location, tags and title;
- The Content Manager web app also provides direct, real-time communication with contributors via audio talkback or via in-screen messaging;
- Allows users to define rules to programmatically manage content and create additional tags to any asset for easy organization.

3.2.3 DaCast

DaCast is an online video platform launched on 2010 that allows enterprises to broadcast and host video content. They also provide API solutions for video content management and video player customization, enabling tailoring of their own services to their customers’ needs.

Thus, DaCast provides a wide range of features that include video hosting on their website (enabling video uploading and sharing), high-definition and low-latency live broadcasting and content monetization. The content monetization is handled either via built-in paywalls or by placing advertisements on videos. The platform is white label⁶ as well, allowing companies to maintain ownership of their own content throughout the streaming process and embed it on their site free from DaCast logos. In a nutshell, some of its main features are:

- Allows connection to webcams or professional cameras to record their live streams;
- Allows the delivery of live streams via CDNs;
- Enables direct publishing to an HTML5 video player;
- Enables content monetization via paywalls or advertising integration;
- Allows users to stream live on Facebook using the social streaming module.

3.3 Comparative Analysis

From the analysis that has been done to the existing platforms during this section, it can be concluded that there is currently no solution that is capable of supporting all the main functionalities that will be provided by the MOGPlay platform as seen in the Table 3.1. The features of the platform that are used to compare all the live streaming solutions presented during this chapter include:

⁶White label - business model in which a product developed by one enterprise can be rebranded by other companies to make it appear as if they had made it.
Live Streaming Applications

- **Mobile streaming** — ability to create live streams on mobile devices (either through a native application or a web app);
- **User’s approximate location** — provide an estimate of the user’s location;
- **Event management** — ability to create events and associate streams to each of them;
- **External publishing** — ability to select a video output for publishing to external broadcast channels;
- **Real-time editing** — ability to mix audio and video for the stream output. This can be done in a multi-viewer panel in which producers look over live streams in real-time;
- **Platform monetization** — enables users of the platform to earn money while using it;
- **Interaction between producers and streamers** — interaction between user managing transmissions (producers) and streamers (i.e., text messages, feedback, etc.);
- **Web-based platform** — all features of the platform are available on web-based applications.

<table>
<thead>
<tr>
<th>Features</th>
<th>Platforms</th>
<th>Facebook Live</th>
<th>Twitch</th>
<th>Livestream</th>
<th>Bambuser</th>
<th>DaCast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile streaming</td>
<td>△ Yes</td>
<td>△ Yes</td>
<td>△ Yes</td>
<td>△ Yes</td>
<td>△ Yes</td>
<td>△ Yes</td>
</tr>
<tr>
<td>User’s approximate location</td>
<td>▽ No</td>
<td>▽ No</td>
<td>▽ No</td>
<td>△ Yes</td>
<td>▽ No</td>
<td>▽ No</td>
</tr>
<tr>
<td>Event management</td>
<td>▽ No</td>
<td>▽ No</td>
<td>△ Yes</td>
<td>▽ No</td>
<td>▽ No</td>
<td>▽ No</td>
</tr>
<tr>
<td>External publishing</td>
<td>▽ No</td>
<td>▽ No</td>
<td>△ Yes</td>
<td>▽ No</td>
<td>▽ No</td>
<td>▽ No</td>
</tr>
<tr>
<td>Real-time editing</td>
<td>▽ No</td>
<td>▽ No</td>
<td>△ Yes</td>
<td>▽ No</td>
<td>▽ No</td>
<td>▽ No</td>
</tr>
<tr>
<td>Platform monetization</td>
<td>▽ No</td>
<td>△ Yes</td>
<td>△ Yes</td>
<td>▽ No</td>
<td>△ Yes</td>
<td>▽ No</td>
</tr>
<tr>
<td>Interaction between producers and streamers</td>
<td>▽ No</td>
<td>▽ No</td>
<td>▽ No</td>
<td>△ Yes</td>
<td>▽ No</td>
<td>▽ No</td>
</tr>
<tr>
<td>Web-based platform</td>
<td>△ Yes</td>
<td>△ Yes</td>
<td>▽ No</td>
<td>▽ No</td>
<td>△ Yes</td>
<td>▽ No</td>
</tr>
</tbody>
</table>

Table 3.1: Overview of live streaming applications

While *Facebook Live* serves a similar purpose to the MOGPlay platform, being widely used in the context of citizen journalism, this platform lacks a crucial component: a GUI meant for producers to filter the live streams captured by the citizens. As such, as seen in the Table 3.1, this platform lacks almost all features that are provided by the MOGPlay platform with the exception of mobile streaming.

As for *Twitch* and *DaCast*, these two platforms are mainly meant for live streaming and video hosting and, as such, these platforms do not offer most of the features that MOGPlay provides for.
Live Streaming Applications

the exception of mobile streaming. *DaCast* also allows platform monetization, providing a good example on how to achieve it in live streaming solutions.

Concerning the other two remaining solutions, the *Livestream* and *Bambuser* platforms are quite complete offering almost all the functionalities required as can be seen in Table 3.1. Nevertheless, they lack some crucial features.

Firstly, the *Bambuser* solution does not allow real-time editing in a multi-viewer panel, a fundamental issue when the objective is to offer a good *Quality of Experience* in live events. In fact, out of all the platforms presented over this chapter, the only one to allow real-time editing of video feeds in event-based scenarios is the *Livestream* platform. This platform not only allows producers to create events and associate streams to those events, it also allows them to visualize streams from multiple sources and select a video output for external publishing.

However, even if the *Livestream* solution provides all the referred features to the producers, it does not provide the system with the user’s approximate location. This feature is fundamental for the editor/producer to select groups of video feeds according to their location. Besides that, the aforementioned solution also does not allow the producer to communicate with the streamers on the event. This would enable the producer to give feedback and indications of what to shoot to the streamers, which would empower them to get the best footage possible from the event.

In fact, there is a lack of platforms dedicated to the communication between the producer in charge of covering the event and the citizens streaming in an area. While most solutions integrate a messaging system in their applications (all presented solutions over this chapter except *DaCast*), most of these systems are not meant for the producer in charge of video switching and the streamers, but only for the broadcaster and their viewers. The only platform which enables communication between the user managing the streams and the streamers is *Bambuser*. Nonetheless, this platform does not provide real-time editing and event management features that are core functionalities in the context of video production. This platform also does not provide any mechanism to monetize the platform, a key issue for businesses of all shapes and sizes.

Lastly, it is also important to mention that most of the solutions that were presented over this chapter are not web-based. Web-based applications provide more accessibility to their users as it is available on almost all platforms/devices. As such, this is another advantage that the MOGPlay platform has when compared to the other solutions that were described previously.

By offering a solution which covers all the content creation process, from the capture and transmission of live content through a mobile app, to the network infrastructure and the web interface where the editor/director can preview a pre-selected set of video feeds and select one for external publishing, the MOGPlay platform stands out from the existing solutions in four main features: its ability to enable the producer to communicate with the streamers, to determine users’ approximate location, to allow real-time editing and to enable the configuration of the solution according to each event’s characteristics.
3.4 Conclusions

Some existing solutions similar to the system to be implemented were presented and discussed during this chapter. The featured solutions were also divided in two major categories for ease of presentation: consumer oriented platforms and business to business (B2B) platforms.

A comparative analysis of the live streaming solutions presented during this chapter was also made. In this analysis, it was concluded that while some of exposed platforms supplied a lot of the features that will also be provided by the MOGPlay project, namely Livestream and Bambuser, there is currently no solution that simultaneously offers all its functionalities.

All in all, it was deduced that there is a lack of web-based platforms that simultaneously provide the following features that will be offered by the MOGPlay platform:

- Communication between journalists and citizens;
- Determination of the user’s approximate location in relation to the event’s area;
- Real-time editing of live videos;
- Configuration of the solution according to each event’s characteristics.
Chapter 4

MOGPlay Platform

In order to better understand the context and motivation to this dissertation’s work, it is important to provide a more detailed overview of the project in which this thesis is inserted in, the MOGPlay platform. This overview is done in this section, providing more information about the main goals of this project.

Besides giving an overview of the goals for the MOGPlay project, it is also important to present an outline of the work that had already been developed by MOG Technologies for the platform before this dissertation. This is necessary because this thesis’ work needs to be merged with the previously developed solution and, as such, it is important to understand its architecture and the features that were already implemented before it.

In addition, it is necessary to detail the challenges that are still not addressed by MOG Technologies’ solution. The identified challenges allow the proper definition of the problem that this thesis tries to solve, introducing the modules that are the focus of this thesis and providing a justification to the research done in the previous chapters.

4.1 Overview

"Cloud-based collaborative content production for live events", or MOGPlay for short, is a project to be developed by a consortium formed by MOG Technologies, INESC TEC, Jornal de Notícias and OSTV. The main objective of this project is to design and develop a cloud-based platform that enables collaborative content production in the coverage of large events.

In order to fully enable this collaboration, four different components need to be designed and developed by the referred consortium, namely:

1. A **cloud-based platform** that enables real-time editing of multiple heterogeneous video streams;
MOGPlay Platform

2. A **mobile web application** for smartphones capable of acquiring high quality video in an event and stream it to the cloud-based platform;

3. A **web-based GUI** capable of receiving multiple streams from the cloud-based platform and dynamically switch different videos inputs based on the producer/director commands;

4. A **Wi-Fi network infrastructure** that is able to support a massive number of users in simultaneous streaming.

When it comes to the division of tasks among the consortium (which can be seen in Figure 4.1), **MOG Technologies** is responsible for the design and development of the cloud-based platform, the mobile web application and the web-based GUI, while **INESC TEC** is tasked with the provision of the Wi-Fi network infrastructure. **Jornal de Notícias** and **OSTV** are the project stakeholders, being responsible for providing its requirements.

![Figure 4.1: High-level overview of the MOGPlay project](image)

In the context of the MOGPlay project, it is assumed that the events’ attendees can be either professionals or common citizens and that they can capture the video streams through the mobile web application represented in the left side of **Figure 4.1**. This mobile web application establishes an HTTP connection to the content provider’s cloud server infrastructure that, in turn, receives and processes the web streams collected from the event venue.

Afterwards, the processed event streams are sent to the web-based GUI. In this interface, the producers in charge of the event are able to examine, browse, filter and select the most relevant streams out of the ones displayed to them. The selected feeds can then be relayed to the Internet, TV, Regie and/or on-site screens. All of this is illustrated on the right-side of **Figure 4.1**.

At the event venue, the event producers should also ensure the existence of an appropriate Wi-Fi coverage infrastructure, with the correct number of different access points to which the
MOGPlay Platform

audience can connect their smartphones to start streaming live video feeds. This is important so that the audience does not rely on their own telecommunications service to stream the content. The Wi-Fi network infrastructure is to be provided by INESC TEC who will study innovative telecommunications systems that provide reliable access to the Internet for a large number of users.

At the time of creation of this document, MOG Technologies has created a prototype for the MOGPlay platform which will be presented below.

4.2 Current Solution

The solution for the MOGPlay project developed by MOG Technologies empowers users of the mobile web application to capture live video and/or audio and transmit it to the cloud in near real-time. The users of the mobile web application are also able to switch the camera that captures the video and mute (or unmute) the audio to be sent in the transmission. These features can be seen at Figure 4.2 which displays the mobile web application of this solution.

Figure 4.2: Mobile web application by MOG Technologies

All the live feeds received in the cloud are then relayed to the web-based GUI, being displayed to the producers. The interface of the MOG Technologies’ solution that enables the real-time management of live streams can be seen at Figure 4.3.
In this interface, it is possible to preview all the streams being captured by the users of the mobile web application in the right side of the page as displayed in Figure 4.3. These stream previews show up and fade on the page in real-time. That is, if a streamer starts recording a live stream, the preview automatically appears on the list of Live stream previews. Likewise, if that same user stops streaming, that stream disappears from the web-based GUI. This is possible due to the usage of Socket.IO that enables real-time, event-based communication.

Socket.IO also enables updates to the image previews of the live streams from time to time. Only image previews are shown on the right side of the web-based GUI so that this page is not overloaded with video content that would slow down the browser if there were a lot of live stream previews being shown.

With the list of stream previews being displayed on the right side of the interface, the producer is able to select one of those streams and drag and drop it into one of the four video players present on the left side of the main panel of the web-based GUI depicted in Figure 4.3.

The platform also allows the publishing of a stream in real-time to external broadcast channels, such as Youtube Live, after selecting a video source in the producer’s GUI. Besides that, the producer can mix video and audio for the external transmission, that is, the producer can select a source for the audio and another for the video to send to the external broadcast channels mentioned previously. This can be seen in Figure 4.3 in which the second stream’s video is selected but the audio that is being sent to external channels belongs to the first transmission displayed in the web-based GUI.
The producer is also able to remove a live stream from the main panel as long that that stream is not being published to external sources. To do that, they only need to press the close button above the video player on the right side and the stream will go back to the list of stream previews.

Now that all features of the current solution to MOGPlay were presented, the architecture behind this project will be explained. An overview of the architecture of the solution developed by MOG Technologies can be seen in Figure 4.4, which contains a representation of the mobile web application, web-based GUI and cloud platform.

The mobile web application is represented in the top left of the figure, while the web-based GUI is illustrated in the top right of the referred figure. As for the cloud platform of the MOGPlay platform, it is composed by three modules that are depicted in the figure as well, to note:

- **The application server**, a Node.js application responsible for managing the whole platform. The application server communicates with the web clients (both the mobile web application and the web-based GUI) using Socket.IO. This technology enables a full-duplex, real-time communication between the application server and the web clients;
MOGPlay Platform

- A media engine composed by Janus (or Janus WebRTC Gateway) and FFmpeg\(^1\), responsible for processing the received web streams. Janus is responsible for handling all incoming WebRTC connections, making the live streams available to FFmpeg. In turn, FFmpeg is responsible for standardizing the video and audio of the received streams as will be explained in more detail below;

- A video server, provided by NGINX\(^2\), which receives the standardized stream from FFmpeg and converts it to a RTMP stream, making it available to the web-based GUI and external broadcast channels, such as Youtube Live and others, through a RTMP link.

On the mobile web application, the live audio and video is captured using WebRTC. When users start recording, the live stream is transmitted to the WebRTC server, the Janus WebRTC Gateway, that is part of MOGPlay’s cloud platform. Then, when Janus receives the user’s transmission, Janus will be able to publish the stream and make the video and audio available for further media processing. After that, the server, the Node.js application, will send a request to the FFmpeg module to start processing the WebRTC media made available by Janus, as shown in Figure 4.4. Janus is also responsible for handling the audio muting or unmuting prompted by the user.

FFmpeg will then standardize the stream’s codecs and video resolution regardless of the original stream’s characteristics. This process, which will convert the codecs of the received stream to predefined codecs and change the video resolution, is also known as transcoding. This is necessary because the resolution and codecs of the original transmission will vary a lot depending on the device sending the transmission. After processing, the output stream from FFmpeg will have a video resolution of 1280 per 720 (width per height) and will use H.264 for the video codec and AAC for the audio codec.

After finishing the media processing, FFmpeg will send the standardized stream to the video server, NGINX, which will convert the received transmission into an RTMP stream. The RTMP transmission can then be relayed to the web-based GUI or to external broadcast channels, such as Youtube Live, television and others as is depicted in Figure 4.4.

It should be noted that the setup of the whole MOGPlay infrastructure was put inside three virtual machine containers: one for the Node.js application, one for the media engine containing Janus and FFmpeg and another for the video server provided by NGINX. These containers make it simpler to deploy these three components of the MOGPlay platform in other machines.

Lastly, it is important to mention that the solution by MOG Technologies does not differentiate the streams received by the platform, meaning that streams are not associated to events and that there is no distinction between the streams coming from citizens and from journalists. There is also no data store associated to the platform.

---

\(^1\)FFmpeg - collection of libraries and tools that process multimedia content such as audio, video, subtitles and other related metadata. Multimedia framework able to decode, encode, transcode and play any type of multimedia.

\(^2\)NGINX - open-source, lightweight web server that provides several configuration options for optimal performance.
4.3 Challenges

Recalling the conclusions taken in chapter 3 that presents an overview of existing live streaming solutions similar to the MOGPlay platform, it was concluded that the features that stand out from this platform include:

- Communication between producers and streamers;
- Determination of the user’s approximate location in relation to the event’s area;
- Real-time editing;
- Configuration of the solution according to each event’s characteristics.

Out of these features, the only one that has already been developed by MOG Technologies is real-time editing as the current solution enables producers to mix video and audio of the received live streams to relay to external broadcast channels.

Therefore, in the context of the MOGPlay platform described previously, there are a wide array of different challenges that can be identified. These challenges need to be addressed not only to fulfill the objectives defined for the project, but also to make this platform stand out from other existing marketing solutions. The challenges that could be identified include:

- Lack of a communication mechanism between event producers and streamers;
- The need to identify the different types of users on the platform;
- The need to monetize the platform;
- Filtering of web streams by event;
- Lack of a data store.

The logic behind these challenges that were the motivation to this thesis’ work will now be presented.

Lack of a communication mechanism between event producers and streamers

In the MOG Technologies’ solution that was described previously, there was no communication mechanism between the event producers and the mobile application users, citizens or journalists.

A communication platform should be present in order to enable the producer to give feedback and indications of what to shoot to the streamers, citizens or journalists. This feedback would provide more motivation to the citizen streamers to keep streaming since they would know that their live stream is being seen by the event producers. All of this would empower the producers to obtain the best footage possible from the event.

The existence of a communication platform would also contribute to making the MOGPlay platform stand out from other existing live streaming solutions.
The need to identify the different types of users on the platform

In the solution described previously, there is no distinction between the different types of users benefiting from the usage of the platform. This differentiation is crucial in the context of this project not only to distinguish streams coming from journalists and from citizens, but also to distinguish journalists with permission to manage the platform (the event producers) from the journalists that only have permission to stream in events. This distinction would allow the platform to restrict access to some areas depending on the type of user that is trying to access them.

The distinction between the streams coming from journalists and from citizens is considered fundamental since it would not only facilitate the stream filtering process for the event producers but would also enable them to give appropriate feedback to the different types of users on the event location in the presence of a communication mechanism.

The need to monetize the platform

The solution developed by MOG Technologies lacks a monetization mechanism. There is a need for this type of mechanism so that it gives some kind of profit for the media companies that will use the MOGPlay platform and make it available for normal citizens. Therefore, it is important for a platform of this kind to include some kind of advertising, in which the owner of the platform can reach advertising agreements with third party companies/brands that want to advertise their products during the event.

Filtering of web streams by event

Another important requirement that is yet to be covered is the filtering of web streams by location so that the streams collected from the event venue can be associated with the corresponding event. With this, the producer would be able to configure the platform to only accept the feeds that are being captured in a certain area that is associated with the event.

As such, this filtering would allow the accomplishment of two of the stand-out features of the MOGPlay platform: to configure the solution according to each event’s characteristics and to determine the user’s approximate location in relation to the event’s area.

Lack of a data store

The existence of a data repository is essential to enable the persistent storage of the platform’s data, including information about the users, events among others. Thus, data stores are crucial to the operation of the whole system as it allows the platform to create, retrieve, update and delete data entities on the system. For example, if a user registers on the platform, a data store would be needed to save the information about the user so that they are able to authenticate into the application at a later time. This storage is also important so that the users do not lose their state when they reload the pages which would improve user experience.
4.4 Problem Definition

My thesis’ work, which was baptized as Operate&Notify, arose from the need to answer all the challenges previously identified. As such, three differing modules were conceived, namely:

- An **authentication** module that addresses the need to identify the different types of users benefiting from the use of the platform. This authentication module should be able to assign roles and permissions to the platform’s users that can be event producers, journalist streamers and citizen streamers;

- A **messaging** module that addresses the need for a communication mechanism on the MOGPlay platform that was described previously. This messaging module should enable event producers to send text or audio messages to one or more streamers (journalists and/or citizens) covering the event;

- A **platform management** module that addresses the need to filter web streams by event and the need for a monetization mechanism. This module should enable the association of areas of interest on the map to events, allowing the filtering of live streams by event on the platform. This is possible only after obtaining the user’s position in relation to the event location. Besides that, this module should enable the event producers to send advertising to the streamers in the context of an event, allowing the platform monetization.

It is also important to mention that a database was conceived for the Operate&Notify solution. All the aforementioned modules communicate with the system’s database to store, retrieve, update and delete the necessary data for the platform to function as intended.

Nonetheless, when it came to the conception and implementation of the solution to the three modules mentioned previously, there were some problems that emerged that conducted to a research on a diversified set of topics.

The most interesting problem that emerged was the selection of the most suitable web-based communication protocols and technologies to use for the messaging system that provide low-latency in the data transmission and real-time communication. It was also important to take the scalability of the communication infrastructure into account. That is, the communication system should enable the event producer to broadcast audio to a variety of journalists in the event venue simultaneously without overloading the event producer’s web client.

Therefore, a series of studies were conducted in chapter 2 with the objective to figure out a solution that solves the problems associated with this thesis in an efficient way. Taking the results of these studies into account, a solution for the three modules mentioned previously was designed and developed, as is presented in the next chapter.
4.5 Conclusions

During this chapter, the MOGPlay platform and its goals were presented in more detail providing more context to this thesis’ work. The main goal of the referred project is to design and develop a cloud-based framework that allows real-time editing of multiple video streams captured in a large event by journalists or common citizens through a web-based mobile application. The specified video streams are simultaneously overlooked and filtered by event producers that will select the most relevant streams in a web-based GUI. The selected feeds can then be relayed to the Internet, TV and/or on-site screens.

The solution to the MOGPlay platform that was developed by MOG Technologies was laid out during this chapter as well, presenting not only its functionalities, but also its architecture. The prototype for the MOGPlay project enables users of the mobile web application to capture live video and audio and to transmit it to the cloud in near real-time. All the live feeds received in the cloud are then relayed to the web-based GUI, being displayed to the producers. The producers are then able to select the most relevant feeds out of the ones being captured and transmit them to external broadcast channels, which include Youtube Live.

The exposition of the MOGPlay prototype allowed for the identification of some existing challenges that still need to be addressed not only to fulfill the objectives defined for the project, but also to make this platform stand out from other existing marketing solutions. The challenges that were exposed during this chapter include: lack of a communication mechanism between producers and streamers, the need to identify the different types of users on the platform, lack of a monetization mechanism, the need to filter live transmissions by event and lack of a data store.

The aforementioned challenges allowed for the conception of the solution to this thesis’ work which was named Operate&Notify that addresses all the problems that were exposed during this chapter. This solution was divided in three modules: authentication module, messaging module and platform management module.
Chapter 5

Operate&Notify Ecosystem

In this chapter, details are provided about the conception and development of Operate&Notify, the work developed during this thesis. The solution is within the scope of the MOGPlay project and was named Operate&Notify because the features provided by this system are mainly concerned with the management (or operation) of the platform and the possibility of notifying users through a messaging system. This work was achieved through the implementation of three different modules: the messaging module, the authentication module and the platform management module.

Firstly, a high-level overview of the proposed solution to this thesis’ work is given, explaining what was achieved with each of the three modules referred previously. After that, the methodology used in the implementation of this project is presented as well.

Furthermore, the requirements of the project implemented in the context of this thesis are defined taking into account the prior exposition of the proposed solution. A use case for this system is also presented in this section to exemplify its usage and how this system might be beneficial to MOGPlay’s users.

After that, the architecture of this project is also presented. In this section, details about the architecture and technologies used to implement the Operate&Notify system are provided so that the reader is able to comprehend the overall structure and organization of the solution.

Thereafter, details about the implementation of the project are also presented. This section presents an walkthrough over the main features of the platform mainly from the user’s point of view. Some technical details are also provided in this section.

Lastly, the results achieved with the solution implemented during this thesis are also addressed and discussed. In this section, the validation of the achieved results is carried out in order to conclude whether the main objectives of this thesis’ work were accomplished with satisfactory results.
5.1 Overview

In the context of the MOGPlay project, a solution named Operate&Notify was designed in an attempt to approach the challenges that were detailed in the previous chapter. Firstly, in order to enable a clear distinction between the features meant for the streamers and the ones meant for the administrators, the platform was divided in two independent web GUls: the **mobile web application** meant for the streamers that can be accessed by the general public and the **web-based GUI** that is reserved for the event producers that are the platform administrators.

Overall, the mobile web application enables its users, that can be journalists or citizens, to record live streams of video and/or audio in an event that will be sent to the MOGPlay’s cloud platform. As for the web-based GUI, this interface enables event producers to look over the received live streams, select video and/or audio for external publishing, send messages to the users and manage the whole platform, being able to administrate events, advertisements and groups. It is important to note that even if these two areas cannot be accessed from one another through the user interface, they still share the same server-side logic behind.

In addition, the proposed solution was divided in three major modules as was mentioned previously: the authentication module that addresses the need to identify the different types of MOGPlay’s users, the messaging module that addresses the need for a communication mechanism and the platform management module that addresses the need to filter web streams by event and the lack of a monetization mechanism on the platform. A database was conceived for the Operate&Notify system as well, addressing the need for a data store. An overview to the three aforementioned modules will be given below, explaining their purpose and providing information about what each of them can achieve.

**Authentication module**

The authentication module allows the identification of MOGPlay’s users as either citizens or journalists so that it is possible to distinguish streams coming from journalists and from citizens. In addition, this module enables the differentiation between journalists with permission to manage the platform, the event producers, from journalists that only have permission to stream in events, the journalist streamers.

Thus, this module enables citizens to stream anonymously (that is, to stream without prior registration) or to create an account and stream as authenticated users, using the mobile web application. This module also empowers journalists to register themselves using the web-based GUI, being that journalists can have either two of the following roles: producer or streamer. A journalist producer (or event producer) is able to access the functionalities provided by the web-based GUI while a journalist streamer is only able to stream as an authenticated journalist using the mobile web application.
**Messaging module**

The messaging module enables the event producers to send text or audio messages to users covering the event, who can be journalists or not, giving them instructions and feedback while using the web-based GUI represented in the top right of Figure 5.1. The aforementioned figure provides an overview of the communication mechanism between producers and streamers that was developed for the MOGPlay platform. The messaging mechanism also enables the producer to send text/audio messages either individually or to a group of users. The messages sent by the producer would then be received in real-time by the mobile application users while they are streaming the event.

![Figure 5.1: Communication mechanism between event producers and streamers](image)

The communication with citizens is essentially made using text messages and simple interactions (e.g. like), while the interactions with journalists are mainly made through audio. It should be noted however that communication through text messages and simple feedback is also allowed for the journalists. This differentiation in the type of data to be sent to the citizens and to the journalists using the mobile web application is illustrated in Figure 5.1.
Platform management module

The event producers have access to a wide range of functionalities that can be accessed in the web-based GUI, empowering them to manage a great variety of content on the platform. Thus, producers have access to a dashboard that enables them to create and manage three components essential to the MOGPlay application: groups, events and advertisements. All these can be considered sub-modules of the platform management module. The solution that allowed overcoming the two challenges associated with this module (filter web streams by event and the need for a platform monetization mechanism) will now be explained.

One way to filter web streams by event is by associating it to an area of interest on the map corresponding to the event venue. Thus, in order to achieve this, the event producers need to be able to create events associated to an area on the map that will determine the area in which the event’s transmissions can be collected.

Then, when an event producer picks a previously created event to manage its transmissions, the live transmissions that show up on the producer’s watcher page will belong to users that are within the area of the event that was previously defined. Note that the watcher page refers to the interface to manage live streams, belonging to the web-based GUI. Therefore, live streams from users that are outside the event area will be filtered out and will not show up on the watcher interface. This is illustrated in Figure 5.2, in which the three transmissions of users within the event’s area appear on the watcher page while the fourth one (from a user that is outside the event area) does not. It should be noted that Event A is the event that was selected by the producer in the illustrated example.

As for the platform monetization issue, one way to achieve it on the MOGPlay platform is to display advertisements on the pages of the citizen streamers. Let us examine the example depicted in Figure 5.3 to understand how this monetization mechanism works. Suppose that a company, e.g. Spotify, is sponsoring a music festival and that this company is willing to pay...
Operate&Notify Ecosystem

ccontent providers to be advertised during the event. The solution to this problem would be to display Spotify advertisements during the event to the citizen streamers in their pages.

Note that advertisements should not be shown to journalists since they work for the content providers and, as such, they are not the intended targets of the advertisements. These ads should also be clickable so that they redirect the users that click on them to the page the company wants to advertise.

Thus, in order to filter web streams by event and to enable the platform monetization, the event producers are able to:

- **Create events.** Each event will be associated to an area in the map determining the area in which the event’s streams can be collected. Events will also be associated to a start and end dates;

- **Select events.** The producers are able to select an ongoing event to filter the streams by the area of the event. That is, if a user is streaming within the area associated with a certain event and if the producer selects that event, then that stream will be displayed in the watcher interface;

- **Create advertisements** in the context of an event. The producer can create advertisements associated to previously created events, providing the advertisement with an image and a web address;

- **Set advertising campaigns** in the context of an event. A producer can create an advertising campaign in an event displaying the event’s advertisements to the streamers streaming in that same event. The producer can also define the targets of the advertisements and set the time interval determining when the advertisements should be updated on the streamers’ pages if there is more than one advertisement associated to an event. The advertising targets can be predefined targets (e.g. all citizen streamers) or previously created groups;
Operate&Notify Ecosystem

- **Create groups of streamers.** These groups can be used to define the targets of the group messages mentioned previously. That is, if a producer creates a group of users and then they select that group when sending messages, those messages will be sent to the users belonging to that group. The created groups can be used to define the targets of advertisements as well;

- **View a listing** of previously created **groups, events and advertisements**;

- **Delete** previously created **groups, events and advertisements**.

### 5.2 Methodology

We will now describe the process that led to the implementation of **Operate&Notify**. The methodology used in the development of this project was the iterative model that can be seen in **Figure 5.4**.

**Figure 5.4: Iterative model [51]**

This model is an iterative process in which an initial, simplified implementation grows in complexity until all the system’s requirements are satisfied. This methodology was used due to its flexibility to changes of the project’s requirements and due to its capability of generating working prototypes early in the software life cycle. A methodology with these characteristics was needed because not all the project’s requirements could be implemented right from the start. This was the case since a prior integration with the **MOG Technologies’** solution was required in order to implement some essential features of the **Operate&Notify** system as is explained in more detail later in this chapter. The steps of this model consist of:

- **Planning and Requirements**: establish the system’s functional and non-functional requirements;

- **Design**: define the software architecture, including the business logic, database models, etc;

- **Implementation**: actual implementation of the project (coding process);
Operate&Notify Ecosystem

- **Verification**: identify and correct issues or bugs found in the developed project;
- **Evaluation**: evaluate the project up to this point and plan the next steps taking into account the conducted evaluation.

Thus, firstly a study was conducted on the problem and on existing solutions that are similar to the MOGPlay platform, which led to the research exposed in the chapters 2 and 3. With this, an analysis and reassessment of the system requirements were carried out.

With the core functionalities of the system properly outlined, some low-fidelity prototypes of the system were developed in the shape of paper sketches of the web application’s pages. This type of prototyping is common practice in the early stages of development, since besides being low-cost and fast to achieve, it allows for the visualization of the system’s functionalities from the user’s point of view.

The initial architecture of the system was also outlined in an attempt to satisfy the project’s requirements and implement the desired user interactions. With these steps concluded, the implementation phase of the project and the ensuing tests of the system took place.

Several versions of the project were implemented iterating over this methodology, which led to further refinements to the requirements, architecture and design of the system along the way. While there were several versions of this project that were implemented using this methodology, we will highlight two versions of the project which have the greatest differences between them: the project before the integration with MOG Technologies’ solution and the project integrated with the main solution.

## 5.3 Requirements Definition

Taking into account the overview that was given to the proposed solution, we will now describe the expected behavior of the system *Operate&Notify* in order to achieve the project’s goals, by listing its functional and non-functional requirements.

A use case for this system is also presented in this section to exemplify its usage and how it might be beneficial to its users. This use case in particular gives emphasis to the messaging module implemented in the context of the *Operate&Notify* system.

### 5.3.1 Functional Requirements

The functional requirements correspond to particular behaviors of the system that can be, directly or not, manipulated by the user. Priorities were given to each requirement to understand how important it is to implement each of them when compared to the rest of the requirements.

In order to better understand the Table 5.1 and Table 5.2 that list the functional requirements, we divided the platform’s users into three major types based on the authentication system that was previously overviewed. These three types are:
Operate&Notify Ecosystem

- **User** – is able to transmit live audio and video to the platform using the mobile web application. A user can be anonymous or may authenticate themselves into the platform. Is also able to receive messages and advertising from the platform. Identified as a citizen on the platform;

- **Journalist** – is able to transmit live audio and video to the platform using the mobile web application. Has the same characteristics as the users, with the exception of not receiving advertising from the platform and not being able to stream anonymously. Identified as a journalist on the platform;

- **Producer** – is allowed to manage all the streams sent by the users and journalists, selecting the most relevant streams for external re-publishing. The producer is also allowed to manage the whole platform, being able to send messages to the streamers (users and journalists) and create and manage groups, events and advertisements. Also identified as a journalist on the platform.

The functional requirements were divided in two main groups according to the component to which they are related. In the first set, displayed in **Table 5.1**, we present the user requirements associated to the mobile web application and **Table 5.2** presents the requirements of the web-based GUI operator.

<table>
<thead>
<tr>
<th>Priority</th>
<th>Actors</th>
<th>Requirement Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>User</td>
<td>Create an account on the mobile web application as a citizen.</td>
</tr>
<tr>
<td>High</td>
<td>User</td>
<td>Authenticate in the mobile web application as a citizen.</td>
</tr>
<tr>
<td>High</td>
<td>Producer &amp; Journalist</td>
<td>Authenticate in the mobile web application as a journalist.</td>
</tr>
<tr>
<td>High</td>
<td>All</td>
<td>Log out of the mobile web application.</td>
</tr>
<tr>
<td>High</td>
<td>All</td>
<td>Transmit a live stream to the platform while authenticated.</td>
</tr>
<tr>
<td>High</td>
<td>User</td>
<td>Transmit a live stream to the platform anonymously.</td>
</tr>
<tr>
<td>High</td>
<td>User</td>
<td>View advertisements on their page.</td>
</tr>
<tr>
<td>High</td>
<td>All</td>
<td>View messages sent to them by the event producers.</td>
</tr>
<tr>
<td>Medium</td>
<td>All</td>
<td>See their own account’s credentials.</td>
</tr>
<tr>
<td>Medium</td>
<td>All</td>
<td>Edit their own account’s credentials.</td>
</tr>
<tr>
<td>Medium</td>
<td>All</td>
<td>Delete their own account.</td>
</tr>
</tbody>
</table>

Table 5.1: Functional requirements for the mobile web application

56
### 5.3.2 Non-Functional Requirements

The non-functional requirements of the developed system will now be exposed. These can be regarded as being the definition of the global characteristics, specifying criteria that can be used
later on to give a generic assessment of the system. Thus, the non-functional requirements of **Operate&Notify** are:

- **Efficiency** – Being a web-based platform, the developed system may easily be overloaded with requests due to its distributed nature. As such, the management of resources should be sturdy enough to handle these types of scenarios;

- **Integrity** – The developed system should be able to ensure the correct access to the content available on the platform. Thus, the platform should prevent any unauthorised access;

- **Scalability** – The platform should be able to handle multiple producers managing the platform and sending messages to the streamers. Furthermore, the system should also be scalable enough to handle multiple journalist streamers connected to the platform, enabling producers to send audio messages to them;

- **Responsiveness** – The pages of the mobile web application must be responsive in different devices (including mobile, tablet and desktop), being that their content should resize with the browser window to fit dynamically as the window is resized;

- **Usability** – As the system’s applications are targeted at a diversified audience of people with varying levels of expertise when it comes to the usage of technologies, it becomes important to consider how to present the information and the system’s features in the most intuitive way possible;

- **Integrability** - In order to facilitate the process of installation and usage of the developed system, it is intended that the system is as easy as possible to integrate in the MOGPlay platform;

- **Performance** - Considering the importance that real-time communication bears for the MOGPlay platform, being one of its major features to allow real-time editing of multiple video streams, it is crucial to ensure that the system’s response time is fast enough so that it does not disrupt the normal operation of the platform.

### 5.3.3 Use Case

The legends of Heavy Metal, *Iron Maiden*, are coming to Portugal on their Farewell Tour at “Estádio do Restelo” and a concert attendee, Cláudia, is highly excited to see them live for the last time, wanting it to be an unforgettable experience. The concert will be outdoors and the tickets are sold out. A music journalist called Rita is also looking forward to get some nice footage from the last concert *Iron Maiden* will ever give in Portugal.

Hugo, the event producer, wants to get the best footage he can get from the concert, so that he can broadcast it to the Internet and people that were not able to attend the concert can enjoy it. He
also wants to broadcast that footage to the local on-site screens so that people that do not have a nice view of the stage can still watch the concert.

Cláudia managed to get a place on the front row after waiting in line for hours on end. She can also see the large on-site screens from where she is. As for Rita, she got a place in the upper row at the other end of the stadium and though she is far away from the stage, she still has a great view of it. She is also away from the speakers so she can listen to the event producer’s audio messages during the event. The disposition of the referred attendees at the event venue can be seen at Figure 5.5.

Ten minutes after the concert started, Cláudia was already having so much fun that she wanted to share that experience with other people around the world. Therefore, she started streaming the concert using the mobile web application of the MOGPlay platform. Hugo saw her footage and thought it was pretty good so he sent her positive feedback represented by a "like" emoji. Then he told her that he would broadcast her stream to the Internet and to the local on-site screens. Cláudia was really happy due to this feedback and kept on trying to get the best footage she could. Soon after that, she saw her own stream on the large on-site screens at the event which made her even more proud of her work.

Shortly after that, Rita received an audio message from Hugo which wanted a different point of view of the event (the audio message came from the MOGPlay platform). After that, Rita started streaming taking into account the instructions given by the event producer. She was also able to see her stream on the on-site screens.

Figure 5.5: Disposition of attendees at event venue
Hugo was overlooking the streams at the event venue of the concert, all the while trying to filter the best feeds out of all that were being received in the Web-Based GUI of the MOGPlay platform, ensuring their transmission to external broadcast channels. He also sent text messages and feedback to the citizens that were streaming the event and dispatched audio messages to the journalists, so that he could send instructions to the streamers on what footage to get. All of this was done in order to get the best footage possible of the concert during the whole duration of the event.

5.4 Architecture

The architecture of the Operate&Notify system will be described in order to present its structural design. We will start by uncovering its logical architecture that provides a high-level overview of the layers comprising the whole system.

As can be seen in Figure 5.6, the Operate&Notify system is based on a client-server architecture and is composed by three layers that communicate with each other: the user interface, middleware and data access layers. As such, the architecture of the developed system is very similar to a three-tier architecture\(^1\).

The user interface represents the presentation tier of the platform, being responsible for displaying the views comprising the two web GUls of the MOGPlay platform, the web-based GUI and the mobile web application. This layer is the one that the end users interact with and communicates with the middleware layer either through API calls in a request-response fashion or through the usage of Socket.IO that enables real-time, bidirectional communication between the web clients and the server as will be explained in more detail later in this chapter. Note that the

\(^1\)Three-tier architecture - client-server architecture comprising three tiers of software, namely the presentation tier, the business tier and the data tier.
The functionalities provided by the platform management module are only available to the end users in the web-based GUI. This is because these features are directed to the platform administrators, the event producers, and as such they are only available in the web GUI that is meant for this type of users.

The middleware layer contains the business logic of the system, being responsible for coordinating the whole application. This layer also provides a RESTful web service that defines the API endpoints that can be called by the mobile web application and web-based GUI to access and manipulate web resources using a set of predefined, stateless operations, namely through GET, POST, PUT and DELETE requests. The middleware layer is responsible for handling the Socket.IO events triggered by the actions of the end users as well. As a whole, the middleware layer implements all modules that comprise the Operate&Notify system while also moving the data between the two surrounding layers, the user interface and data access layers.

As for the data access layer, this tier is responsible for storing all data provided by the middleware layer in a database. It also retrieves previously stored data and sends it back to the middleware component.

Further details about each of the system’s layers will be provided following a bottom-up approach. Thus, we will firstly present the data access layer, supplying information about the database incorporated in this system and the entities that are saved in it. Afterwards, we will provide details about the key technologies used in the application server that comprises the middleware layer.

Subsequently, the communication architecture of the platform will be presented explaining how the web clients and the application server exchange messages in real-time and what technologies were used to achieve that.

Implementation details about some of the key components/features of the project will be provided as well, including the platform’s authentication, the system’s security and the retrieval of the user’s location. Lastly, details about the deployment of the whole system will be supplied.

### 5.4.1 Data Access Layer

A database was designed and implemented in order to store the data needed to manage the platform and to implement the authentication and messaging systems. The software used to create the database was PostgreSQL which is a relational database management system. This software is not only free and open-source, but also widely used due to its flexibility, high performance and easiness to maintain.

As seen in Figure 5.7, the system’s database stores the credentials of the registered users and of the anonymous streamers. Each user is associated with a set of permissions that grants them or not access to certain functionalities, to note:

- Streaming as a citizen in the mobile web application;
- Streaming as a journalist in the mobile web application;
• Managing the whole platform in the web-based GUI.

Depending on their permissions, users have access to different areas on the platform. Users can also assume two different roles: the role of a streamer and the role of a producer. Users with the role of a streamer are labeled as either citizens or journalists, while producers are associated to a set of messaging options available on their page as is shown later in this chapter at Figure 5.23. Users can also belong to groups that are created by the producer.

All text messages are saved in the database as well so that the required information of each message is associated to the message history of the user that received it. Besides saving the ids of the users sending and receiving the message, the date in which it was sent and its content are also stored.
Events created by the producer are also stored in the database saving the information provided in the *Create event page* seen in Figure 5.25. Users can be associated to an event as the event managers (if `manage` is true) when a producer selects an event to manage its streams or as the event streamers (if `manage` is false) when users are streaming in the event area. Note that not only events can have multiple streamers (that is, can be associated to multiple users) but users can stream in multiple events as well.

Advertisements created by producers are also saved in the database with the information provided in the *Create Advertisement* page depicted in Figure 5.27. Events can also be associated to advertisements.

From the ads associated to an event, an advertisement can be chosen to be displayed in the pages of the event’s streamers. Advertisements are displayed to the event’s streamers when advertising campaigns are created for the desired event. The settings for the advertising campaign are saved in the *Chosen Advertisement* entity. The saved settings include whether to target all citizen streamers (if `target_all` is true) or only non-visible streamers (if `target_all` is false) in the event, the update interval which determines when the chosen advertisement is refreshed and the timestamp of the last update. Note that non-visible streamers refers to users that are making live streams that are not in the main panel of the producers’ watcher pages, that is, they are users that are either not streaming or that are not making interesting live streams. Advertising campaigns can also be targeted to specific groups of users in the context of events.

The communication with the data access layer is implemented on top of a repository pattern which allows us to abstract the handling of the database operations for each domain object. This enables the separation of the business logic from the database logic which in turn reduces duplicated code for the database operations and makes it easier to switch data store at a later point in time.

### 5.4.2 Application Server

The application server of the *Operate&Notify* system was implemented using *Node.js*, being responsible for providing the core functions of the platform. Besides providing a RESTful API and handling the *Socket.io* communications, the *Node.js* application also acts as the platform’s web server, being in charge of serving the static files comprising the mobile web application and the web-based GUI. The package manager *npm* was used to manage the application dependencies.

The *Express.js* framework was used along with *Node.js* in order to simplify the development of the API endpoints and to deliver the web pages of the MOGPlay platform. A template engine called *EJS* (Embedded *JavaScript* templates) is used as well to avoid replicated HTML code in multiple pages and to facilitate the design of dynamic HTML pages.

The *node-postgres* library was used to make queries to the *PostgreSQL* database from the *Node.js* application. *node-postgres* provides features such as connection pooling, a design/software pattern that allows multiple queries using the same connection to the database, which made it simpler to interface with the *PostgreSQL* database from *Node.js*.
5.4.3 Communication Architecture

When it came to the conception and implementation of the solution to the messaging module, there was an interesting challenge that emerged: the selection of the most suitable web-based communication protocols and technologies to use for the communication infrastructure of the messaging system. The communication infrastructure should not only provide low-latency in the data transmission and real-time communication, but should also be scalable. That is, the communication infrastructure should enable the event producer to broadcast audio to a variety of journalists in the event venue simultaneously without overloading the event producer’s web client.

Having in mind the study conducted in the chapter 2 in order to solve this problem, a solution for the communication mechanism was projected as it is represented in Figure 5.8. This solution enables event producers to send audio and/or text messages to the users of the mobile web application, while using the web-based GUI.

As depicted in Figure 5.8, WebRTC was chosen as the technology for sending audio messages from the event producers using the web-based GUI to streamers using the mobile web application. In fact, in chapter 2, an overview of WebRTC and some of its alternatives was made. Despite the protocols presented in this chapter being viable alternatives when it comes to live streaming (namely RTMP, HLS and MPEG-DASH), these technologies were not suitable choices in the context of this work. This is because the MOGPlay platform must enable bidirectional real-time communication between web clients, as the mobile application users will need to be able to transmit live video and audio to the event producers while the producers need to be able to send text and/or audio messages to the streamers. The WebRTC alternatives described do not support these features as they only allow the distribution of live video and audio from the broadcaster to the viewers.

![Figure 5.8: Communication architecture](image)

64
Another pivotal point to consider in the choice of WebRTC is that this technology enables low-latency transmission of both audio and video, which is something that is much harder to achieve with HLS or MPEG-DASH which are usually associated with high-latency. As such, WebRTC was deemed as the most suitable option in the context of this project.

Considering that audio broadcast is a requirement for the system to be implemented, a media server must be used alongside WebRTC, implementing a peer-to-server architecture rather than a peer-to-peer architecture. As was concluded in chapter 2, Janus WebRTC gateway currently offers the sturdiest solution to media servers, being an open source solution that enables the implementation of a scalable media server.

As depicted in Figure 5.8, Janus simplifies the implementation of the signaling mechanism that is needed in order to communicate between WebRTC clients. Another advantage of Janus when compared to other media server solutions is that Janus was already being used to capture live video and audio from the mobile web application in the prototype developed by MOG Technologies. In fact, in this solution, the Janus server was set up inside a virtual machine container and the Janus’ JavaScript library was already being used in the streamers’ pages. This not only made it easier to implement the audio messages, but also did not overload the streamers’ pages with more dependencies. As such, Janus was used along with WebRTC in order to solve the scalability issue mentioned previously.

As for the technology used for the exchange of text messages, Socket.IO was deemed as the most suitable technology to use for this purpose. Socket.IO is a Node.js library that enables bidirectional communication between the web clients (represented in Figure 5.8 as the web-based GUI and mobile web application) and the application server. This choice was made because Socket.IO uses WebSockets as the main transport protocol to deliver messages. Recalling the conclusions taken in chapter 2, WebSockets was deemed to be the most appropriate protocol to use to build scalable real-time applications, characteristics that are highly desired in the context of the messaging system of Operate&Notify.

Other reasons for choosing this technology is that it provides a wide range of features essential to this work such as reliable connections, auto-connection support among others. Besides that, using Socket.IO made it easier to implement the text messages functionality. It was also compatible with the solution by MOG Technologies, since Socket.IO was already being used to enable real-time updates to the pages of the web clients of the MOGPlay platform. Furthermore, using this technology alongside WebRTC provided a secondary channel for the transmission of different types of data. Therefore, even in the case of WebRTC failure, the text messaging mechanism would still work.

In conclusion, in order to implement the communication infrastructure for the messaging and notification system, WebRTC was used along with Socket.IO to enable the transmission of audio and text messages. Janus was also used as the WebRTC gateway as can be seen in Figure 5.8, acting as the media server and simplifying the implementation of the signaling mechanism.
5.4.4 Additional Implementation Details

We will now present additional implementation details about some of the key components in the platform like the system’s authentication and security. We will also provide additional details about the technologies and methodologies used to retrieve the user’s approximate location, a core feature of the Operate&Notify solution.

Authentication

The Web Storage API was used with fallback to cookies for older browsers to identify each user that employs the mobile web application and the web-based GUI on the client-side of the MOG-Play platform. These technologies are used when a user logs into the system successfully to determine which user is currently logged-in on a browser and to restrict access to certain areas on the platform.

In the current iteration of the Operate&Notify system, in order to expedite development and to facilitate prototyping of new features, user permissions are being handled on the client-side of the platform. When a user does not have permission to access a certain page on the platform, they are redirected to the index page of the mobile web application or the web-based GUI. Although this has the advantage of hastening the validation of the new functionalities, it also has a drawback of being vulnerable to client-side user manipulation.

It is expected, at a later point in time, to move the handling of permissions to the server layer to thwart or minimize client-side tampering. A possible approach would be to employ a proven library such as Passport.js that provides multiple authentication methods, enabling external authentication through other websites such as Facebook.

Security

Some security measures were taken in order to ensure the protection and integrity of the system’s data. The main security measures that were taken during the development of the Operate&Notify system will now be described. The two security measures to be presented were taken at the data access layer while executing queries and while saving sensitive data to the database.

Firstly, passing parameters to the database queries directly using string concatenating was avoided as this would create unwanted SQL injection vulnerabilities. As such, the database queries were parameterized using node-postgres that passes the provided parameters to the PostgreSQL server which in turn sanitizes queries from any SQL injection.

An encryption method was also used to store the users’ passwords safely. The encryption method that was applied is called bcrypt which is a cryptographic function that incorporates salt in the hashing process. Salt is additional random data that helps making each password hash unique, given that its main purpose is to avoid rainbow table attacks. This method is used to encrypt the passwords stored in the database.

\[^2\] Rainbow Table - precomputed hashes lookup table, usually used to crack hashed passwords.
password so that the hashed version of the password is saved on the database when users create an account. When an existing user tries to authenticate in the platform, bcrypt is used again to compare the given password to the user’s hashed password.

Now we will talk about two security measures that were taken at the client-level of the platform, ensuring the integrity of the web pages and the security of the connections to the web pages.

The messages sent by the event producers are escaped from HTML entities so that messages containing unsafe characters such as HTML tags do not disarrange the web pages of the platform. That is, if unsafe characters were not escaped, then the producers could send HTML tags in their messages which could easily disarrange the structure of the platform’s web pages.

Lastly, it is also important to mention that secure connections (HTTPS) are used to serve the web pages of the system and to provide the application’s API endpoints. Secure connections are used to provide a guarantee regarding the website’s authentication and to avoid eavesdropping on the requests between the web clients and server.

User’s Location

In the streamers’ pages of the mobile web application, the user’s location is obtained using the Geolocation API, a native API to the browsers, which allows the retrieval of the user’s location after they give their permission. When the user’s location is obtained, a request to the server API is made to get the events associated to the user, checking if the user is within the area of valid, ongoing events. The request to get the user’s location is called from time to time to update the events associated to the user. The Geolocation API is more accurate for devices with GPS as is the case for most smartphones. It should also be noted that in most browsers, the Geolocation API only works in secure contexts (i.e., while using HTTPS connections).

5.4.5 Deployment

The setup of the whole infrastructure of the platform including the Node.js application and the PostgreSQL database was put inside a container, an Alpine Linux virtual machine. This was done so that the deployment of the system could easily be done anywhere, since all dependencies are encapsulated within the virtual machine and no additional configurations are necessary.

As such, in order to run the MOGPlay platform, three containers need to be run: the virtual machine with the core of the application that was described previously and two virtual machines from the MOG Technologies’ solution that were described in the previous chapter. One of those virtual machines contains the setup of the media engine (Janus and FFmpeg) and the other contains the video server (provided by NGINX).
5.5 Prototype Implementation

The development of the Operate&Notify system was divided in two major phases as was mentioned previously: the implementation before the integration with the main project of the MOG-Play platform and after the integration.

The integration with the main project would have an impact on the implementation of some features of the Operate&Notify system, in particular the features comprising the whole messaging module and the filtering of web streams by event. This was the case because these features were dependent on some functionalities that were already provided by MOG Technologies’ solution, namely: the ability to record live streams using the mobile web application and the ability to manage live streams on the web-based GUI.

Considering that the messaging module required integration with the main project, the focus of the development in the first phase of the project was in the authentication module and the platform management module. Thus, most of the features comprising these modules were already implemented at the end of this phase.

Taking this into consideration, the interfaces comprising the authentication and platform management modules were not modified before and after the integration with the main project. The interfaces that underwent changes due to this integration were the producer’s watcher page and the streamer’s page. One of the initial versions of the watcher page before the integration with the main project can be seen in Figure 5.9.

---

Figure 5.9: Initial version of the watcher interface
To facilitate the development of the messaging system, a prototype for the messaging module was implemented using Socket.IO during the first phase of the project. This prototype allowed event producers to send broadcast messages to all connected streamers using the watcher page as seen in Figure 5.9. This page also enabled the producer to access their dashboard by pressing the Dashboard button depicted in Figure 5.9. In the dashboard, the producers would be able to manage the whole platform.

The user interface of the watcher page underwent a major refactoring between the two phases of the project. This was because the components of the Operate&Notify system were integrated in the watcher page of the solution by MOG Technologies. The final version of the watcher page can be seen later in this chapter in Figure 5.20.

As for the streamer’s page, the components of the solution by MOG Technologies were integrated in the streamer’s page of the Operate&Notify solution. As such, this page did not change much between the two phases of the project as can be seen in Figure 5.10 that depicts the before and after of the streamer’s page. The page seen on the left of the aforementioned figure is the streamer’s page before the integration and the one on the right is the same page after the integration. It should be noted that the authentication system was already implemented in the first phase of the project, given that the streamer’s page shown in Figure 5.10 belongs to an authenticated journalist.

Figure 5.10: Streamer page before and after integration with the main project

This implementation process made it easier to integrate with the main project of the MOGPlay platform and achieve the final version of the project. Thus, at the end of the second phase, the
features comprising all three modules of the *Operate&Notify* system were implemented with success.

We will now present the final version of the MOGPlay platform that includes the *Operate&Notify* system developed in the context of this thesis. As mentioned earlier, this application was divided in two independent web GUIs: the mobile web application meant for the streamers and the web-based GUI meant for the event producers. Below, the main functionalities provided by these two web GUIs will be exposed by explaining the three modules that were implemented for the *Operate&Notify* system: the authentication module, the messaging module and the platform management module.

### 5.5.1 Authentication Module

In order to easily distinguish between journalists and citizens, an authentication system for the different types of users of the MOGPlay application was designed. This authentication system is represented in Figure 5.11 and allows the identification of users on the platform. This module also enables the platform to restrict access to certain areas on the website depending on whether the users trying to authenticate in the referred areas have permission to access it or not. This system is also important to identify the different producers when they send messages to the streamers to provide a stronger sense of proximity between the streamers and producers.

![Figure 5.11: Overview of MOGPlay’s user types](image)

This authentication system allows users of the MOGPlay platform to be identified as either journalists or citizens enabling a clear distinction between these two types of users. Journalists can also have two different roles on the platform, that is, a journalist can either have the permissions of a producer or those of a streamer.

There are three areas hidden behind an authentication system in the MOGPlay platform as depicted in Figure 5.11 and this system can be accessed from the home pages of the two web GUIs. Firstly, we will talk about the authentication system on the mobile web application.
The home page of the mobile web application seen in Figure 5.12 allows reaching the streamer’s page either through an authentication system differentiated for citizens and journalists or anonymously. The streamer’s page can be seen in Figure 5.13, which highlights the elements that were added to the page due to the Operate&Notify system.

The general area refers to a streamer’s page on the mobile web application that can be accessed by any user that is registered on the platform. The journalist area refers to a streamer’s page that can only be accessed by journalists, given that users labeled as citizens cannot access it. Note that journalists cannot access the platform without credentials.

It should be noted that users identified as citizens can only authenticate in the general area of the mobile web application. However, contrarily to what happens with journalists that can only use the platform when authenticated, citizens can be anonymous as well. That is, users accessing the streamer’s page with no credentials will be identified as citizens on the MOGPlay platform.

The referred streamer’s page provides different functionalities depending on whether the user is a journalist or a citizen. If the user is a citizen then they will be able to record live streams, receive text messages and view advertisements but they will not be able to receive audio messages from the event producers. Otherwise, if the user is a journalist, then they will be able to record live streams, receive text and audio messages from the event producers but they will not view any advertising on their page. The streamer’s page shown in Figure 5.13 is one of a citizen, since an advertisement is shown on that page.
The referred streamer’s page also provides different features depending on whether the user is authenticated or not. If the user is authenticated, they also have access to a menu which gives them access to a dashboard in which they can update their credentials, delete their profile or log out. Authenticated users also have access to the functionalities that were listed before depending on their type (either citizen or journalist). If the user is anonymous then they only have access to the features provided to citizens. This difference between authenticated and anonymous users can be seen in Figure 5.14.

![Figure 5.14: Difference between authenticated and anonymous users](image)

The authentication page which enables all registered users to login into the application is displayed in Figure 5.15, while the authentication page that provides access only for the journalists can be seen in Figure 5.16. These two authentication pages can be accessed from the home page of the mobile web application (Figure 5.12).

![Figure 5.15: General area login](image) ![Figure 5.16: Journalist area login](image)
Note that while a button leading to the registration page is available on the general area login page (Figure 5.15) allowing users of the mobile application to register on this page, this button is not visible on the login page reserved for journalists. This is because the registration page for the journalists is not available on the mobile web application, being hidden from the general public. This should be the case so that citizens using the mobile web application are not able to create accounts as journalists.

Overall, as is illustrated in the Figure 5.17, the users of the mobile web application (the streamers), both journalists and citizens, benefit from an authentication system that gives them access to the streamer’s page in three different ways, namely:

- As an **authenticated user** (general access with credentials). Authenticated users need to register and authenticate before accessing the streamer’s page. Any user with credentials can stream as an authenticated user. The registration can be done in a page that can be accessed by everyone and users registered in this page are identified as citizens;

- As an **anonymous user** (general access without credentials). Anonymous users are identified as citizens on the platform and they only need to access the streamer’s page to create a live stream. This page can also be accessed by everyone;

- As an **authenticated journalist** (restricted access). The registration page for the journalists is hidden from the general public and is only available on the web-based GUI, preventing
citizens from accessing it. Journalist streamers are identified as journalists in the platform and advertising is not shown to them on their pages.

Now we will talk about the authentication system on the web-based GUI. All functionalities available on the web-based GUI are hidden behind an authentication system that enables access to the admin area referred in Figure 5.11 for users with permissions for it. Note that the admin area refers to the producer’s watcher page and dashboard. The login page that leads to the admin area on login success is the home page of the web-based GUI seen at Figure 5.18.

![Figure 5.18: Web-based GUI’s home page](image)

![Figure 5.19: Journalist registration](image)

The registration page for the journalists is also available on the web-based GUI as seen at Figure 5.19, being that all users registering in that page will be identified as journalists. This page can be accessed from the login page available on the web-based GUI that is seen at Figure 5.18. As such, journalists registering with the permission to Manage Streams ticked on will have the role of producers (journalist producers) while journalists registering with the referred checkbox ticked off will have the role of streamers (journalist streamers).

A journalist producer, mainly referred to as event producer throughout this document, is able to access all functionalities available on the MOGPlay platform with their credentials, either on the web-based GUI or on the mobile web application. As such, an event producer is able to authenticate in the administration area available on the web-based GUI through the login page seen at Figure 5.18. They are also able to authenticate in the journalist area and general area available on the mobile web application that were mentioned previously, as is represented in Figure 5.11.
As for the journalist streamers, they can access all areas available on the mobile web application, being able to authenticate in the area that is meant for them, the journalist area, and the general area. However, these journalists are not able to authenticate in the administration area and, as such, are not able to manage the whole platform.

5.5.2 Messaging Module

The messaging system of the Operate&Notify solution was implemented in the watcher page of the web-based GUI and in the streamer’s page of the mobile web application. The watcher page is the interface that enables the producer to send messages and can be reached by authenticating into the platform with producer credentials through the producer’s login page. The streamer’s page of the mobile web application display the messages sent by the producer. The watcher interface can be seen in Figure 5.20.

Before the authenticated producer is able to send messages to users sending the live streams, firstly the producer needs to choose one of the stream previews that are listed on the right side of the page as displayed in Figure 5.20. After selecting the stream, the producer needs to drag and drop the preview of the selected stream into one of the four video players present in the main panel of the watcher page.

When the producer drops the preview into one of the available players, an overlay will show up over the video displaying information about the user that is creating that live stream and showing
all messages that were previously sent to that user as seen in Figure 5.21. The user’s information that is displayed on the referred overlay includes their username, their id and their type that can be either citizen or journalist. If the user creating the stream is not authenticated, then their username will be displayed as Anonymous by default as is exemplified on the overlay over the third video player in Figure 5.20.

The individual messaging options represented in Figure 5.21 will be enabled when the producer drops the live stream in the video player above (they are disabled if no video is playing). These messaging options allow the sending of text and/or audio messages to the user creating the live stream as seen in Figure 5.21. A shortcut for sending simple feedback messages in the shape of likes is also available.

To send a text message, the producer only needs to select the text input, write a message and press the Enter key or the Send button that becomes available when the corresponding text input is selected. The text message that was sent will then show up on the message history of the target in real-time on the watcher page, that is, the producer does not need to reload the page for the message history to update. The text message also appears simultaneously on the page of the streamer to which the message was sent to. This was achieved using Socket.IO that enables bidirectional communication based on events and, in this case, these events are triggered by the producer sending the text messages.

To send an individual audio message, the producer needs to press the button with a microphone icon to start recording and then press the button with a stop button to stop recording the audio
message, as can be seen in Figure 5.21. The target journalist streamer will then be able to listen to the audio message sent by the producer. This was achieved by making use of the Janus’ JavaScript library. It should also be noted that audio messages are only available for journalists and if the user sending the live stream is a citizen, the audio message option will not be available.

As could also be seen in Figure 5.20, the event producer is able to send group audio messages and group text messages. The group audio messages are only sent to the journalists available on the main panel (up to four journalists). To send a group audio message, the producer only needs to press the Broadcast button available in the main panel of the producer as exemplified in Figure 5.22. A timer will also be shown displaying for how long the producer has been transmitting audio.

In addition, the producer has access to a set of customisation options for the messaging system. To access the messaging options, the producer needs to open the Messaging Options modal as seen in Figure 5.23.

![Figure 5.22: Group audio message](image)

![Figure 5.23: Producer’s messaging options](image)
In this modal, the producer can enable or disable individual messages, being that when individual messages are disabled, the individual messaging options are hidden from the watcher page as depicted in Figure 5.24. Note that the individual messaging options are shown by default. As for the group text messages, the producer is able to choose between two different modes of group messages, the broadcast mode and selection mode as is depicted in Figure 5.23.

The broadcast mode allows producers to send text messages to the streamers present in the main panel of the watcher interface. For example, if there are two users creating live streams and those two live streams are playing in two of the video players in the main panel of the watcher page, then when the producer sends a group text message in this mode, the message is sent at the same time to those two users. This is the group messages mode that is selected by default.

If the selection mode is selected, check boxes will appear associated to all streams that show up on the watcher interface, including the streams appearing only as previews on the right sidebar of the page as seen in Figure 5.24. The selected check boxes determine the targets of the group text messages sent by the producer. It is also to important to mention that this mode not only allows the producer to select the targets of the messages by hand, but also allows them to select previously created groups so that the users belonging to those groups are automatically selected on their page.

Figure 5.24: Selection mode (group messages)

The group text messages sent by the producer also appear in real-time on the message history of the message targets, both in the watcher page and in the targeted streamers’ pages. The group audio message is received by all targets simultaneously as well.
5.5.3 Platform Management Module

The event producers have access to a dashboard that can be accessed from the watcher webpage that enables them to create and manage three components: events, advertisements and groups. These three components are the sub-modules of the platform management module. We will now describe in detail the functionalities provided by these three sub-modules.

Events

An event producer is able to create events in the Create Event page on their dashboard, by giving them a name, start date, end date and by associating them to an area on the map as depicted in Figure 5.25. The start date defines the starting point in which the event starts and the selected date/time can be any that is before the end date. The end date defines the date in which the event ends and cannot be a past date.

As for the event location, the Google Maps API was used in order to make it easier for the producers to define an area of interest on the map. In order to select the event location, firstly the producer needs to enter the name of a place in the Location input depicted in Figure 5.25 which will give auto-complete suggestions to the user from the Places API of Google Maps.

When the producer selects one location out of the suggestions that were given to them, a marker centered on the selected location will appear on the map with a highlighted circle around it. This circle delimits the area in which the live streams of that event will be collected. The
producer can then drag the margins of the circle and define the area of the event as they so desire. They can also drag the marker around the map and drop it in another desired location on the map.

When the producer presses the button Create Event, the event is created with success if the provided fields are valid. The created event can then be seen on the Manage Events page shown in Figure 5.26 which lists all events that were created beforehand.

![Manage Events Page](image)

**Figure 5.26: Manage events page**

On this page, the producer can select an event in order to configure the watcher page to filter web streams by event as was illustrated in Figure 5.2. A producer can also delete an event from the platform and create an advertising campaign in the context of an event as long that there is at least one advertisement associated to the event as will be explained below.

It should also be noted that if there is no ongoing event available and/or if a streamer is outside the areas of interest associated to previously created events, then the users of the mobile web application will be unable to create live streams and the button to start recording will be disabled until the streamer is within the area of a valid, ongoing event. The error that shows up when the user is unable to stream is displayed in Figure A.4.

**Advertisements**

In order to enable the platform monetization, event producers are able to create advertisements in the context of events, by giving them a name (usually the name of the sponsor) and by associating
them with an image and web address as can be seen in Figure 5.27. The producer has to associate the advertisement to one or more events as well in order to specify the events in which the advertisement will be displayed.

The image provided during the advertisement creation is saved on the file system of the project. If the provided credentials are valid, then when the producer presses the button Create Advertisement, the advertisement is created with success. The created advertisement can then be seen on the Manage Advertisements page which lists all advertisements that were created beforehand as depicted in Figure A.17. This page also enables producers to delete advertisements and activate or deactivate an advertisement. That is, the event producers are able to disable a certain advertisement without deleting them so that it is not displayed on the pages of the targets.

Note that creating the advertisement does not automatically display it in the streamers’ pages. In order to display it, the producer needs to first set an advertising campaign in the desired event. This can be achieved by going to the Manage Events page that was previously shown in Figure 5.26 and by opening a modal that allows event producers to manage the advertising campaign of the desired event as seen in Figure 5.28.
The tab of the advertising campaign modal that is opened by default is the tab displaying information about the campaign (Campaign Information). This tab allows producers to see the advertisements that are associated to the event. If there is an advertising campaign already set for the event, then it shows the id of the advertisement that is currently being displayed on the page of the event streamers and shows the time left until the next update to the advertisement. Note that the timer shown on this modal is updated in real-time every second. This modal’s tab also allows the producer to unset the current advertising campaign if there is one already set for the event.
Operate&Notify Ecosystem

The other tab of this modal, named *Set Advertising Campaign*, enables producers to create an advertising campaign in the context of the event, allowing the definition of the advertising targets, that is, the streamers that will see advertisements on their pages. The advertising targets of the campaign can be either predefined groups (*All Citizen Streamers* or *Non-Visible Citizen Streamers*) or custom target groups.

*All Citizen Streamers* are users labeled as citizens streaming in the event. *Non-Visible Citizen Streamers* are users with the labeling "citizen" that are making live streams in the event that are not in the main panel of the producers’ watcher pages. The custom target groups are groups that can be created by the producer in the *Create Group* page as will be explained below. Note that the created groups are only listed if the *Target Groups* option is selected.

The producer also needs to specify the time interval between updates to the advertisements in minutes or hours. That is, if there is more than one active advertisement associated to an event, then this time interval determines for how much time each advertisement will be shown. It should be noted that the advertisement that is displayed on every update is different from the previous one and is chosen randomly (if there is more than one active advertisement for the event).

When the advertising campaign is successfully set, the targets of the advertising will be shown an image along with the advertisement banner on their page that redirects to the advertisement URL when clicked on. For reference, the advertisements on the streamers’ pages are updated in real-time. Thus, the streamers do not need to reload their page to view the newly set advertisement when they are one of the advertising targets. In addition, the streamers do not need to reload their page for the advertisements to disappear on their page when the event’s advertising campaign is not set or the streamer is not one of the campaign’s targets.

**Groups**

The producers are also capable of creating groups of users in their dashboard as shown in Figure 5.30. In this page, all users of the platform are listed and the producer is able to select them to create a group.

There are three different types of filters available to help with the user selection process, namely:

- Filter by labeling. Allows producers to pick users by their type (either citizen or journalist);
- Filter by visibility. Allows producers to pick users by their visibility on the main panel of the producers;
- Filter by events. Allows producers to pick users streaming in the selected events.
When the producer presses the Create Group button, a pop-up modal will appear confirming the group creation as can be seen in Figure A.18 and the producer will have to give a name to the new group. If the given name is valid, then the group is created with success.

The created group will then be listed in the Manage Groups page that can be seen in Figure A.19. In this page, all previously created groups are listed and information about the users that belong to those groups is also displayed. The producer is able to delete previously created groups as well on this interface.

Note that the created group will now appear in the advertising targets (if the target groups option is selected) as depicted in Figure 5.29 and in the messaging options (when the selection mode is picked) as was shown in Figure 5.23.

### 5.6 Results

We will now describe the results that were achieved at the end of this work, comparing them with what was initially expected from it. An evaluation of the obtained results is also carried out in order to assess whether they were satisfactory or not.

When it comes to the fulfillment of the project’s functional requirements that were described in Table 5.1 and Table 5.2, the level of satisfaction regarding the accomplished results is very high, being that all the established requirements were implemented and tested with positive results throughout the course of this thesis’ work.
In fact, the developed system addresses all challenges that were previously identified, providing the following features overall:

- Enables users of the mobile web application to stream as citizens (authenticated or anonymous) or as journalists, providing a clear distinction between citizen and journalist streamers;
- Allows users of the mobile web application to create an account as a citizen streamer and users of the web-based GUI to create an account as a journalist with the role of producer or streamer, assigning different roles to the different types of users on the platform;
- Provides a messaging system to the event producers in which they can send text and/or audio messages to one or more streamers (citizens or journalists), providing a communication mechanism between event producers and streamers;
- Enables event producers to create events associated to areas of interest that they can select to filter web streams by event on the platform;
- Enables event producers to create advertisements and set advertising campaigns in the context of events, enabling the platform monetization.

As for the project’s non-functional requirements, it is more difficult to evaluate each of them in an objective way due to their generic nature. However, we will try to provide an assessment of the following characteristics that were intended for the system: efficiency, integrity, scalability, responsiveness, usability, integrability and performance.

As for the platform’s efficiency, the system has not yet been overloaded with requests, so it cannot be said conclusively that the system is highly efficient. However, the smaller scale tests that have been carried out up to this date suggest that this objective has been successfully met.

The integrity of the platform is ensured by the authentication mechanism previously described and by the assignment of permissions to the different types of users of the MOGPlay application. This allows the platform to restrict access to certain areas on the website depending on whether the users trying to access the referred areas have permission to access those areas or not, which guarantees correct access to the platform’s content.

The scalability of the platform is guaranteed through the usage of Janus WebRTC Gateway that enables audio broadcast to a large number of users. The referred technology provides a media server solution that is able of processing media better than the web browser would be able to. Thus, Janus allows producers to send audio messages to a wide variety of journalist streamers.

The responsiveness of the web pages comprising the mobile web application is also guaranteed by the platform. Special care was taken when designing these pages so that they could be properly rendered in a wide variety of devices. The mobile web application was also tested in a lot of different devices and adjustments were made to ensure that the content of the pages was properly displayed.
When it comes to the usability of the platform, particular caution was taken when designing the web pages of the mobile web application and the web-based GUI, always taking into account the user experience it provides to the end users of the platform. In fact, tooltips and placeholders were placed on many elements of the pages to improve the navigation flow while also making the platform usage more intuitive to its users. As such, considering the simplicity of the user interactions it provides, we can say that the level of usability of the platform is satisfactory, enabling an adequate user experience.

Taking into account that the Operate&Notify system was successfully incorporated in the main project of the MOGPlay platform, it is safe to say that the developed system is integratable. As mentioned previously, some of the project’s requirements implied a prior integration with the main project and, as such, this thesis’ work was merged with the main project as soon as all the other features that did not depend on this integration were successfully implemented. In addition, modular programming was applied during the whole development of the project. This facilitated the integration process since the addition of the main project’s features did not affect the already existing components of the Operate&Notify system.

Some tests were performed using Google Chrome’s developer tools for performance and the test results were satisfying on all pages of the platform, with regards to response time, rendering performance and dependencies’ payload. Particular care was given when including external dependencies so as to reduce the pages’ loading time. Therefore all libraries were placed, minified, either locally or in CDNs. Thus, it can be concluded that the performance objective has been met as well.

Considering that the outlined objectives were met within the prescribed deadline, MOG Technologies decided to show the project’s results to the project’s stakeholders and consortium members, namely Jornal de Notícias, OSTV and INESC TEC. As such, a meeting was held on the 21st of May in Jornal de Notícias’ facilities in which the results of the project were presented. In this meeting, a presentation was made about the Operate&Notify system and a video of the main functionalities provided by this system was shown as well. At a later date, on the 12th of June, another meeting was held at MOG Technologies’ facilities in which a demo of this thesis’ project was shown to the consortium members. The meeting participants found the new features of the MOGPlay platform to be promising, opening the doors to new directions in the product’s development.
Chapter 6

Conclusions and Future Work

Over the recent years live streaming has become a popular trend among events’ attendees, allowing them to share their own experiences of the event with people around the world. Realizing that this might be an opportunity for common citizens to contribute for the coverage of large events, MOG Technologies started a project called "MOGPlay" in a partnership with INESC TEC, Jornal de Notícias and OSTV with the aim to design and develop a cloud-based framework that allows real-time editing of multiple video streams captured in a large event by the users using their smartphones.

The specified video streams can be captured by both professionals and common citizens through a mobile web application, being overlooked and filtered by event producers that will select the most relevant streams in a web-based GUI. The selected feed can then be relayed to the Internet, TV and/or on-site screens.

The solution to the MOGPlay platform that was developed by MOG Technologies was laid out during this dissertation in chapter 4. The prototype developed by MOG Technologies does not differentiate the streams received by the platform, meaning that streams are not associated to events and that there is no distinction between the streams coming from citizens and from journalists. There is also no data store associated to the platform.

Taking into account the features already provided by MOG Technologies’s solution, different challenges could be identified and associated to this project. Some of those challenges also derived from the necessity of making this project stand out from other existing live streaming platforms that were presented and analyzed in chapter 3. At the end of the referred chapter, a comparative analysis was done in order to provide an overview of all the applications that were discussed while comparing them to the MOGPlay platform. This allowed us to identify some features of this platform that can provide added value when compared to other existing marketing solutions.

The challenges that could be identified taking into account the designated stand out features and the prototype developed by MOG Technologies included:
Conclusions and Future Work

- The need of a communication mechanism that enables event producers to communicate with the streamers, so that producers are able to give feedback and indications of what to do next to the mobile application users;

- The need to identify the different types of users on the platform so as to distinguish streams coming from journalists and from citizens and to assign differing permissions to the platform’s users;

- The need to monetize the platform so that the content providers benefit monetarily from its use;

- The need to filter web streams by event so that the streams collected from the event venue can be associated with the corresponding event;

- The need for a data store so that the platform is able to save, retrieve, update and delete data entities on the system like users, events among others.

Therefore, as part of the objectives defined for the MOGPlay project, a solution named Operate&Notify was designed in an attempt to approach the challenges mentioned previously. Firstly, in order to enable a clear distinction between the features meant for the streamers and the ones meant for the administrators, the platform was divided in two independent web GUIs: the mobile web application meant for the streamers that can be accessed by the general public and the web-based GUI that is reserved for the event producers.

The system was also divided in three major modules when it comes to its features: the authentication module, the messaging module and the platform management module. A brief overview of what was achieved with these three modules will be given below:

- The authentication module addresses the need to identify the different types of users using the platform. To implement this module, a differentiated authentication system was implemented for the different types of users: one for the event producers on the web-based GUI and two others available on the mobile web application meant for the journalist streamers and for the citizen streamers. Citizens are also able to stream anonymously in the platform;

- The messaging module addresses the need for a communication mechanism on the MOG-Play platform that enables event producers to send text or audio messages to one or more streamers (journalists and/or citizens) covering the event. This messaging system provides a means of communication between journalists and citizens so that citizens can effectively contribute to the coverage of large events with the help of event producers;

- The platform management module addresses the need to filter web streams by event and the need for a monetization mechanism. This module is composed by three sub-modules including groups, events and advertisements that event producers are able to manage in the web-based GUI. As such, this module enables event producers to create events associated to areas of interest that they can select to filter web streams by event on the platform. It also
Conclusions and Future Work

enables producers to create advertisements and set advertising campaigns in the context of events, enabling the platform monetization. This module also allows the creation of groups that can be used to select the targets of the group text messages or to select the advertising targets.

When it came to the conception and implementation of the solution to the messaging module, there was an interesting problem that emerged: the selection of the most suitable web-based communication protocols and technologies to use for the communication infrastructure of the messaging system.

From the research that was conducted during chapter 2 to solve this problem, it could be concluded that WebRTC currently offers the most complete solution to real-time communications over the Web as it allows building peer-to-peer applications for voice calling, video chat and P2P file sharing using native browser APIs. It could also be concluded WebSockets provide the most suitable solution for client-server communications to build scalable, real-time applications. As such, WebRTC and Socket.IO, a technology that uses WebSockets as its main transport protocol, were the selected technologies for the implementation of the messaging system. Janus was also used to facilitate the implementation of audio message broadcast and to facilitate the WebRTC implementation.

Lastly, it is important to mention that all of the main objectives of this thesis’ work were fulfilled with satisfactory results, being that all the project’s functional and non-functional requirements were accomplished. The Operate&Notify system was successfully integrated in the main project of the MOGPlay platform as well. The results of this thesis’ work were shown to the other consortium members of the project, namely Jornal de Notícias, OSTV and INESC TEC, which found the new features presented to them to be promising.

6.1 Future Work

While the results achieved with this thesis’ work were satisfactory with its results being shown to the project’s stakeholders, there are still some aspects of the platform that need to be improved. These improvements can be made not only to provide a higher level of security and robustness to the platform but also to make the MOGPlay platform stand out even more from other live streaming solutions. Thus, there are various aspects that could be considered in future follow-ups of this thesis’ work, some of which that have been suggested by the project’s stakeholders, namely:

- Improvements to the authentication system. In the current iteration of the Operate&Notify system, in order to expedite development and to facilitate prototyping of new features, user permissions of the authentication module are being handled on the client-side of the platform. As such, it is expected, at a later point in time, to move the handling of user permissions to the server layer to thwart or minimize client-side tampering.
Conclusions and Future Work

- Conceive a gratification system to the citizen users of the mobile web application. Identifying citizen streamers on the platform through the authentication system opens the doors for the possibility of rewarding users that create valuable contributions to the coverage of events. For example, this system could be conceived so as to allow the rating of the streams created by citizens. Then, if a user creates a certain number of contributions and their average stream rating is high, users could be rewarded either monetarily (through virtual coins such as Bitcoins) or through free goods at events such as beverages. This would highly increase the motivation of citizens to contribute to the coverage of events;

- Display statistics of advertising campaigns. Displaying statistics about the advertising campaigns would enable event producers to determine the reach of the advertisements shown to the users of the mobile web application. This, in turn, would allow the incorporation of internet advertising models, such as pay-per-click or cost per impression. For example, displaying the count of clicks which were made on the advertisements would enable a pay-per-click model in which advertising sponsors would pay to the content provider for each click on the ad.

- Display video advertisements. Enabling producers to send video advertisements to the mobile application users would make advertising campaigns more varied and flexible. This would make sponsors more willing to have their ads shown on the MOGPlay platform, which would further enable the platform monetization.

- Enable event producers to edit previously created events, groups and advertisements. At the moment, producers are only able to create, view and delete the three referred components. However it is also important to empower them to modify the platform’s content, so that all CRUD operations are available for events, groups and advertisements.
Appendix A

Prototype

A.1 Mobile Web Application

Figure A.1: General area registration (citizen registration)

Figure A.2: Journalist login page (permission denied error)
Figure A.3: Citizen login page (login success)

Figure A.4: Streamer page (invalid event error)

Figure A.5: Streamer page (anonymous user)

Figure A.6: Streamer page menu (authenticated journalist)
Prototype

Figure A.7: Edit profile page

Figure A.8: Dashboard menu

Figure A.9: Delete profile modal

Figure A.10: Change password page
Prototype

Figure A.11: Streamer page (landscape mode)

Figure A.12: Streamer page (hidden content)
A.2 Web-based GUI

Figure A.13: Watcher page (individual text message)

Figure A.14: Publish stream modal
Prototype

Figure A.15: Messaging options modal

Figure A.16: Dashboard home page
Prototype

Figure A.17: Manage advertisements page

Figure A.18: Create group modal
Prototype

Figure A.19: Manage groups page
References


REFERENCES


REFERENCES


REFERENCES


