reTHINK framework evaluation through application development

Bernardo Marques Graça

Dissertation submitted to obtain the Master Degree in Telecommunications and Informatics Engineering

Supervisors: Prof. Ricardo Jorge Feliciano Lopes Pereira
Prof. Ricardo Jorge Fernandes Chaves

Examination Committee
Chairperson: Prof. Rui Jorge Morais Tomaz Valadas
Supervisor: Prof. Ricardo Jorge Feliciano Lopes Pereira
Member of the Committee: Prof. Rui António Dos Santos Cruz

October 2017
Acknowledgments

Firstly, I would like to thank my advisors, Professor Ricardo Pereira and Professor Ricardo Chaves, for their help, guidance and support over the past year. Also, for letting me participate in the reTHINK project and get this incredible experience with a project of this dimension before beginning a new phase in my life.

To my family, girlfriend and friends for all the encouragement and advice throughout these five years of university. Without them, it would have been much harder to get here and accomplish all this. It was a great journey.

Lisbon, October 2017
Bernardo Marques Graça
Abstract

With the goal of providing an alternative model to the currently dominant walled garden communication networks, the reTHINK project provides a new framework for Peer-to-Peer web application development that handles governance, security and identity management for the registered users. In addition, this European project aims to offer a global, open and shared identity system, enabling dynamic trusted relationships among distributed applications without relying on communication protocols.

The reTHINK project has recently finished and a stable version is already available to be used by programmers to develop their own applications. Currently, an evaluation taking into account different perspectives is needed.

This document presents an methodology with the goal of evaluating the reTHINK framework from the point of view of ease of developing applications when compared with traditional technologies, testing the benefits and costs in terms of complexity. This methodology was followed to perform this evaluation. To accomplish this goal, two versions of the same web application was developed and an evaluation performed considering three different points of view: development, applications, and the impact on users and developers.

With this work, we highlight the main advantages and disadvantages of using the reTHINK framework for application development and provide alternatives to improve this framework. To conclude, and given the presented evaluation, we provide the most complete set of recommendations to make the reTHINK framework more usable, easier to use and, consequently, better accepted and easily adopted by the developer community.

Keywords: framework, application, reTHINK, evaluation, web, development
Resumo

Com o objetivo de fornecer um modelo alternativo às atuais redes de comunicação baseadas em ecosistemas fechados, o projeto reTHINK providencia uma nova framework para o desenvolvimento de aplicações Peer-to-Peer, lidando não só com a segurança, mas também com a gestão de identidades dos utilizadores. É de salientar ainda que, este projeto oferece um sistema de identidades aberto e partilhado, permitindo a criação de relações dinâmicas e confiáveis entre aplicações distribuídas sem dependerem dos protocolos de comunicação.

Este projeto terminou recentemente e já está disponível uma versão para que os programadores desenvolvam as suas aplicações. Neste momento, é necessário que seja feita uma avaliação tendo em conta diversas perspetivas.

Este documento apresenta então uma metodologia que tem como objetivo avaliar a framework do reTHINK do ponto de vista de facilidade no desenvolvimento de aplicações quando comparado com outras tecnologias, testando os benefícios e custos em termos de complexidade. Para atingir este objetivo, desenvolvemos duas versões da mesma aplicação proposta e realizamos uma avaliação tendo em conta três pontos de vista diferentes: desenvolvimento, aplicações e, o impacto sobre os utilizadores e programadores.

Com este trabalho, realçamos as principais vantagens e desvantagens de utilizar a framework do reTHINK para o desenvolvimento de aplicações, justificando cada uma delas e fornecendo soluções para que a framework possa ser melhorada. Por fim e, considerando todas as conclusões, fornecemos as principais recomendações para que a framework do reTHINK possa ser mais fácil de usar e, consequentemente, melhor aceite pela comunidade de programadores.

**Palavras-chave:** framework, aplicação, reTHINK, avaliação, web, desenvolvimento
# Contents

Acknowledgments iii

Abstract v

Resumo vii

List of Figures xiv

List of Tables xv

Acronyms xvii

1 Introduction 1
  1.1 Motivation ................................................................. 2
  1.2 Proposed Work .......................................................... 3
  1.3 Outline ................................................................. 4

2 Background 5
  2.1 Web Application Development ........................................ 5
  2.2 Communication .......................................................... 6
    2.2.1 Hypertext Transfer Protocol ..................................... 6
    2.2.2 Remote Procedure Call .......................................... 6
    2.2.3 WebSocket ...................................................... 7
    2.2.4 WebRTC ......................................................... 7
  2.3 Security in Web Development ......................................... 8
    2.3.1 Sandboxing Mechanism ......................................... 8
    2.3.2 Cryptography .................................................. 8
    2.3.3 Secure Communication ......................................... 9
    2.3.4 Validation .................................................... 9
    2.3.5 Typical Web Vulnerabilities .................................. 10
  2.4 Web based Identification and Authentication ....................... 10
    2.4.1 HTTP based Authentication .................................... 11
    2.4.2 Cookies ........................................................ 12
### 2.4.3 Local Password based Authentication .......................... 12
### 2.4.4 Digital Certificates ............................................. 12
### 2.4.5 Token based Authentication ...................................... 13

### 2.5 Chapter Summary .................................................. 15

### 3 Related Work ....................................................... 16

#### 3.1 Framework Selection ............................................. 16

#### 3.2 Framework Evaluation ........................................... 17

#### 3.3 The reTHINK framework .......................................... 17

##### 3.3.1 Architecture .................................................. 18

##### 3.3.2 Usage Example ............................................... 20

#### 3.4 Full-Stack Frameworks .......................................... 21

##### 3.4.1 Sails.js ........................................................ 21

##### 3.4.2 Ruby on Rails ............................................... 22

##### 3.4.3 Laravel ........................................................ 22

##### 3.4.4 CakePHP ...................................................... 23

##### 3.4.5 Django ........................................................ 23

##### 3.4.6 Tornado ........................................................ 24

#### 3.5 Non-Full-Stack Frameworks ...................................... 26

##### 3.5.1 Express ......................................................... 26

##### 3.5.2 Spark Java .................................................... 26

##### 3.5.3 Zend .......................................................... 27

#### 3.6 WebRTC Frameworks .............................................. 27

##### 3.6.1 PeerJS ........................................................ 28

##### 3.6.2 EasyRTC ....................................................... 28

##### 3.6.3 SimpleWebRTC ............................................... 29

#### 3.7 UI Frameworks .................................................... 29

##### 3.7.1 Bootstrap ...................................................... 30

##### 3.7.2 Semantic UI .................................................. 31

##### 3.7.3 Materialize ................................................... 31

#### 3.8 Chapter Summary .................................................. 32

### 4 Proposed Evaluation Methodology ................................. 33

#### 4.1 Methodology ....................................................... 33

#### 4.2 Main Target Features ............................................. 34

#### 4.3 Application Proposal .............................................. 35

##### 4.3.1 Requirements ................................................ 35

#### 4.4 Architecture ....................................................... 36

##### 4.4.1 Supported by the State of the Art ............................ 36

##### 4.4.2 Supported by the reTHINK framework ....................... 37
4.5 Chapter Summary .......................................................... 38

5 Video Conference Application supported by the reTHINK framework 40
  5.1 Software Choices ....................................................... 40
  5.2 Implementation ....................................................... 41
    5.2.1 Authentication of Users ....................................... 41
    5.2.2 Chat Rooms ..................................................... 42
    5.2.3 Video Calls ..................................................... 45
  5.3 Chapter Summary ..................................................... 46

6 Video Conference Application supported by the State of the Art 47
  6.1 Software Choices ..................................................... 47
  6.2 Implementation ..................................................... 48
    6.2.1 Authentication of Users ....................................... 49
    6.2.2 Chat Rooms ..................................................... 50
    6.2.3 Video Calls ..................................................... 51
  6.3 Chapter Summary ..................................................... 52

7 Evaluation and Comparison Analysis ........................................... 53
  7.1 Development Evaluation ............................................ 53
    7.1.1 Documentation .................................................. 53
    7.1.2 Programming Effort ............................................. 54
    7.1.3 Development Time .............................................. 55
    7.1.4 Ease of Installation and Deployment ............................ 56
  7.2 Applications Evaluation ............................................. 58
    7.2.1 Performance Tests .............................................. 58
    7.2.2 Portability Tests ................................................. 62
  7.3 Users and Developers Evaluation .................................... 63
    7.3.1 Tests with Users ............................................... 64
    7.3.2 Tests with Developers .......................................... 66
  7.4 Overall Analysis and Recommendations .............................. 71
  7.5 Chapter Summary ..................................................... 72

8 Conclusions ............................................................... 73
  8.1 Summary ............................................................. 73
  8.2 Future Work .......................................................... 74

A Database systems .......................................................... 76
  A.1 SQL vs NoSQL ......................................................... 76
B Developed Applications

B.1 Application supported by the reTHINK Framework
   B.1.1 User Profile
   B.1.2 Chat Room Invitation

B.2 Application supported by the State of the Art
   B.2.1 User Profile
   B.2.2 Chat Room Invitation

C Evaluation
   C.1 Applications Evaluation

Bibliography
List of Figures

2.1 HTTP basic authentication scheme using HTTPS ........................................... 11
2.2 HTTP digest authentication scheme ................................................................. 11
2.3 Component interaction in OAuth 2.0 ................................................................. 14
2.4 Component interaction in OpenID Connect ....................................................... 15
3.1 reTHINK framework components ................................................................. 19
3.2 reTHINK framework usage example ............................................................... 20
4.1 Architecture supported by the State of the Art ................................................. 37
4.2 Architecture supported by the reTHINK framework ........................................ 38
5.1 Authentication form provided by the reTHINK framework ............................... 41
5.2 User identity object supported by the reTHINK framework ............................. 42
5.3 Reporter - Observer model .............................................................................. 43
5.4 Chat room example ......................................................................................... 43
5.5 Child Data Object example ............................................................................. 45
5.6 Call supported by the reTHINK framework ..................................................... 46
6.1 Developed authentication form ....................................................................... 49
6.2 Account data model ......................................................................................... 49
6.3 Peer configuration ............................................................................................ 50
6.4 Peer network ................................................................................................... 51
6.5 Call using PeerJS framework ........................................................................... 52
7.1 Deployment tests of the application supported by the reTHINK framework ...... 59
7.2 Web page loading test of the application supported by the State of the Art ...... 60
7.3 Deployment test of both developed applications ............................................... 60
7.4 User search tests ............................................................................................. 61
7.5 Message delivery tests ................................................................................... 62
7.6 Longest loading time ....................................................................................... 64
7.7 Relevance of the loading time ........................................................................ 64
7.8 Preferred authentication form .......................................................................... 65
7.9 Develop a custom authentication form ............................................................ 65
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.10</td>
<td>Fastest user search</td>
<td>65</td>
</tr>
<tr>
<td>7.11</td>
<td>Fastest messaging service</td>
<td>65</td>
</tr>
<tr>
<td>7.12</td>
<td>Using an Hyperty</td>
<td>67</td>
</tr>
<tr>
<td>7.13</td>
<td>Knowledge to use an Hyperty</td>
<td>67</td>
</tr>
<tr>
<td>7.14</td>
<td>Multiple Hyperties</td>
<td>68</td>
</tr>
<tr>
<td>7.15</td>
<td>Knowledge to combine Hyperties</td>
<td>68</td>
</tr>
<tr>
<td>7.16</td>
<td>Setup the reTHINK environment</td>
<td>69</td>
</tr>
<tr>
<td>7.17</td>
<td>Change the Hyperty functionality</td>
<td>69</td>
</tr>
<tr>
<td>B.1</td>
<td>Application user profile supported by the reTHINK framework</td>
<td>77</td>
</tr>
<tr>
<td>B.2</td>
<td>Invitation to join the chat room supported by the reTHINK framework</td>
<td>77</td>
</tr>
<tr>
<td>B.3</td>
<td>Application user profile supported by the State of the Art</td>
<td>78</td>
</tr>
<tr>
<td>B.4</td>
<td>Invitation to join the chat room supported by the State of the Art</td>
<td>78</td>
</tr>
<tr>
<td>C.1</td>
<td>Global discovery tests</td>
<td>79</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Full-stack frameworks</td>
<td>25</td>
</tr>
<tr>
<td>3.2</td>
<td>Non-full-stack frameworks</td>
<td>27</td>
</tr>
<tr>
<td>3.3</td>
<td>WebRTC frameworks</td>
<td>29</td>
</tr>
<tr>
<td>3.4</td>
<td>UI frameworks</td>
<td>31</td>
</tr>
<tr>
<td>6.1</td>
<td>Additional integrated modules</td>
<td>48</td>
</tr>
<tr>
<td>7.1</td>
<td>Documentation analysis</td>
<td>54</td>
</tr>
<tr>
<td>7.2</td>
<td>Programming effort overview</td>
<td>55</td>
</tr>
<tr>
<td>7.3</td>
<td>Development time breakdown</td>
<td>56</td>
</tr>
<tr>
<td>7.4</td>
<td>Development time breakdown without Hyperty development</td>
<td>56</td>
</tr>
<tr>
<td>7.5</td>
<td>Ease of installation metrics</td>
<td>57</td>
</tr>
<tr>
<td>7.6</td>
<td>Deployment tests overview</td>
<td>59</td>
</tr>
<tr>
<td>7.7</td>
<td>User search tests overview</td>
<td>60</td>
</tr>
<tr>
<td>7.8</td>
<td>Message delivery tests overview</td>
<td>62</td>
</tr>
<tr>
<td>7.9</td>
<td>Portability tests overview</td>
<td>63</td>
</tr>
<tr>
<td>A.1</td>
<td>SQL vs NoSQL Databases</td>
<td>76</td>
</tr>
</tbody>
</table>
List of Acronyms

AJAX  Asynchronous JavaScript and XML
AP    Authenticating Party
API   Application Programming Interface
CA    Certificate Authority
CDF   Cumulative Distribution Function
CSP   Communication Service Provider
CSRF  Cross-Site Request Forgery
DHT   Distributed Hash Table
DTLS  Datagram Transport Layer Security
GUID  Global User Identifier
HMAC  Hash Message Authentication Codes
HTML  Hypertext Markup Language
HTTP  Hypertext Transfer Protocol
HTTPS Hypertext Transfer Protocol Secure
ICE   Interactive Connectivity Establishment
IDM   Identity Module
IdP   Identity Provider
IP    Internet Protocol
IPC   Interprocess Communication
JSON  JavaScript Object Notation
MD5   Message Digest 5
MVC   Model-View-Controller
NAT   Network Address Translation
OP    OpenID Identity Provider
ORM   Object Relational Mapper
OTT  Over The Top

OWASP  Open Web Application Security Project

P2P  Peer-to-Peer

PKI  Public Key Infrastructure

QoS  Quality of Service

REST  Representational State Transfer

RP  Relying Party

RPC  Remote Procedure Call

RTCWeb  Real-Time Communication in Web-Browsers

RTT  Round-Trip Time

SP  Service Provider

SQL  Structured Query Language

SSL  Secure Socket Layer

SSO  Single Sign-On

STS  Strict Transport Security

STUN  Session Traversal Utilities for NAT

TCP  Transmission Control Protocol

TLS  Transport Layer Security

TURN  Traversal Using Relays around NAT

UI  User Interface

URL  Uniform Resource Locator

XML  eXtensible Markup Language

XSS  Cross-Site Scripting

WebRTC  Web Real-Time Communication

WWW  World Wide Web
Chapter 1

Introduction

The telecommunications industry is evolving at a dizzying rate. In fact, today, the traditional telecommunications based services, like voice telephony, are losing importance at a fast pace. In addition, users are looking for solutions over the Internet that bypass the traditional operator’s distribution. Solutions different from those normally offered by operators, e.g. a solution over the Internet that allows the access to different media content.

To fill the demand, Over The Top (OTT) services started to grow drastically, especially after the introduction of the Web Real-Time Communication (WebRTC) standard [1], that enables real-time communication capabilities between browsers for audio, video and data exchange over Peer-to-Peer (P2P) connections. Because of these new services, new opportunities started to appear for P2P content distribution, e.g. file sharing. This resulted in the emergence of new services and applications such as WhatsApp\(^1\) and Netflix\(^2\).

However, most of these services and applications, operate on a closed system, also known as walled garden [2]. Nowadays, we have Skype from Microsoft, Facetime from Apple, and Duo from Google. Each big company has its own equivalent service, each stuck in its own bubble. These services may be great, but can be problematic since they cause vendor lock-in and limit the portability of user identity and data [3]. In addition, we have the social factor. Users have friends that use all types of applications and services. To interact with them, users cannot use their favorite application or service because they are not interoperable. Instead, users are forced to use the same applications, making it extremely difficult for new applications to succeed, on a competitive and already crowded market.

Therefore, these new services and applications are not exactly what we imagined during the dream years when the Internet was being built, where everyone could participate in equality. Fortunately, there is an emerging movement to bring the Internet back to the original vision. A project that is trying to be a part of this movement, is the European founded reTHINK\(^3\) project that proposes a new framework for web application development.

The reTHINK project gives a greater emphasis to communications, providing solutions to manage

\(^{1}\text{https://www.whatsapp.com , last accessed October 13th, 2017}\)
\(^{2}\text{https://www.netflix.com , last accessed October 13th, 2017}\)
\(^{3}\text{https://reTHINK-project.eu , last accessed July 2nd, 2017}\)
real-time communication capabilities, aiming to create dynamic trusted relationships between distributed applications and provide implicit interoperability. To achieve such goals, the reTHINK project involves the creation of dynamic web-based services named Hyperties that can be described as software modules dynamically deployed in web runtime environments on end-user devices. The communication between different Hyperties from different Service Providers (SPs) is based on the Protocol-On-The-Fly (ProtoFly) [4] concept, which allows the use of standard network protocols without the need to modify them or to create new ones. The ProtoFly pulls code on-demand to dynamically select, load and instantiate the most appropriate protocol stack (e.g. JavaScript file) during runtime. A in depth view on the reTHINK framework is provided in Chapter 3, explaining in detail each component framework and the interconnection between them.

The ultimate goal of reTHINK is to provide an alternative model to the currently dominant walled garden communication networks and, at the same time, empower users with the choice and the management of their private data and identities. Programmers are encouraged to developed new communication enabled applications using the reTHINK framework. As a consequence, this will allow users from different reTHINK enabled applications to communicate with each other without the need to use the same standard network protocols.

1.1 Motivation

The motivation behind this work was the emergence of the reTHINK project. The reTHINK project was recently finished and a stable version is already available to be used by programmers to develop their own applications. Like similar projects, the reTHINK project was developed by a closed group of programmers, which creates the need for an external evaluation of the project. Additionally, this evaluation should not be focused only on the framework components and its features.

Although, we have contributed to the development of the reTHINK project along this year, we tried to be more as external developers. The aim with this, was to identify possible limitations and maximize its impact.

With this in mind, this work consists of an evaluation of the reTHINK project, focusing on the ease in developing applications when compared to traditional technologies, testing the benefits and the costs in terms of complexity. Our greatest difficulty is to conceive an appropriate evaluation methodology, which takes into account the programmers' point of view and that within the scarce human resources available could determine the platform's competitiveness in relation to the existing ones.

The presented evaluation focuses on the development of applications and in the applications, itself (e.g. performance). Although, it is the intended for this work, it raised some subjectivity issues because most of the collected data will be obtain from our programming skills, knowledge and the technologies that we choose. To minimize this problem, we decided to add a third point of view, the impact on users and developers. The idea is through programming events give the opportunity to the users and developers to try the developed applications and to use the reTHINK framework to develop their own applications. In exchange, we could gather a good amount of users’ and external developers’ feedback.
that will be very important for this evaluation.

\section*{1.2 Proposed Work}

The aim with this thesis was to present an evaluation of the reTHINK framework, focusing on the ease in developing applications when compared to traditional technologies, testing the benefits and the costs in terms of complexity. To achieve this goal, we had to design the most appropriate evaluation methodology that in this case it was divided into four parts.

Firstly, we defined a web application capable to evaluate the reTHINK framework by covering the most important use cases targeted by the framework. In Chapter 4 is provided a full description of the proposed application as well as the main target features of the reTHINK framework to be used, tested and evaluated.

To be able to compare the development using the reTHINK framework with the development using traditional technologies, we proposed the implementation of two versions of the same application. One version was implemented taking advantage only of the features provided by the reTHINK framework, while the other version was developed using traditional technologies (e.g. frameworks, libraries, databases), that a typical programmer would use to develop this type of web application. All the used technologies for the second version of the application were selected taking into account the study carried out in Chapter 2 and 3. Along each implementation, we tried to understand what are the advantages, disadvantages and limitations of each technology.

Completing these implementations, we applied an evaluation composed by three different perspectives. Firstly, we did an evaluation focused on the development of each application using different metrics such as documentation quality, programming effort and development time. The aim with this first evaluation was to gather the main advantages and disadvantages that a developer gets by using such technologies, especially with the reTHINK framework.

After that, we proceed with an evaluation over the developed applications using two different types of tests. First, we did some performance tests to collect the efficiency of the most important features present in both develop applications. It was important to test those features because they are directly related with the used technologies. Afterwards, we did some portability tests to see the behavior of both developed applications in different environments.

The last evaluation carried out was the users’ and external developers’ evaluation that makes this whole evaluation more robust and not so subjective. The users’ evaluation was focused on the developed applications, using a survey to collect some initial feedback about particular aspects detected along the implementation and evaluation of both applications. The developers’ evaluation was a much more complex evaluation, where experienced developers had the opportunity to use the reTHINK framework to develop their own web applications. For this, several code challenges were used and developers were asked to answer a detailed survey, with the aim of gathering feedback about the reTHINK framework considering their experience in this area. For both evaluations, a statistical analysis was conducted over the answers, allowing us to extract several conclusions about the reTHINK framework.
At the end, and considering the results of this evaluation, we present an overall analysis and some recommendations to the reTHINK project. We think that following these recommendations will make the framework more usable and efficiency in a near future.

1.3 Outline

This document describes the research and the work developed, and it is organized as follows:

- **Chapter 1** presents the motivation and proposed work;
- **Chapter 2** presents the background on web application development;
- **Chapter 3** describes the previous work in the field;
- **Chapter 4** describes the proposed evaluation methodology;
- **Chapter 5** describes the implementation of the application supported by the reTHINK framework;
- **Chapter 6** describes the implementation of the application supported by the technologies chosen considering the Chapter 2 and 3;
- **Chapter 7** describes the evaluation carried out and the corresponding results;
- **Chapter 8** summarizes the work developed and proposes future work.
Chapter 2

Background

This Chapter provides some background about web application development as well as on important aspects that a developer needs to consider while developing this type of applications. Firstly, we present an overview about web application development, focusing on the technologies used for client and server side coding, scripting and programming. Then, communication protocols such as Hypertext Transfer Protocol (HTTP), Remote Procedure Call (RPC), WebSocket and WebRTC are reviewed. Thirdly, we discuss some security issues that normally web applications face and some protections existing vulnerabilities. To conclude, we introduce some identification and authentication mechanisms used nowadays in web applications, such as cookies, digital certificates and token based authentication.

2.1 Web Application Development

Usually, a web application is a client-server software in which the client or User Interface (UI) runs in a browser. Common web applications include webmail, online retail sales, online auctions, wikis, instant messaging services and many others. To implement these services correctly, the developer should consider some important aspects such as programming language, communication protocol, security issues and authentication of users. There are two main categories of coding, scripting and programming for creating web applications: client side and server side.

Client side: In the client side, the code is executed or interpreted by browsers and it is generally viewable by any visitor. There are some common technologies used for client side coding, scripting and programming such as Hypertext Markup Language (HTML), CSS, JavaScript and jQuery. Additionally, the programmer can also use frameworks, like Bootstrap¹, to help him with the creation and design of the user interface. Those frameworks provide functionalities that are already tested and working properly, allowing to speed up the development process.

Server side: In the server side, the code is executed or interpreted by the web server and is not

¹http://getbootstrap.com , last accessed July 3rd, 2017
viewable by any visitor. There are some common technologies used for server side coding, scripting and programming such as PHP, Ruby, Python, JavaScript and Java. Also in this case, the programmer can use a framework, like CakePHP\textsuperscript{2}, to help in the implementation of certain features such as the authentication of users.

2.2 Communication

One important aspect that the programmer should consider when dealing with web applications is the communication protocol used to exchange data, generally between the client (browser) and the web server. Web applications use HTTP that already provides communication capabilities to the applications. However, the programmer has available other protocols based on different rules and specifications such as RPC, WebSocket and WebRTC.

2.2.1 Hypertext Transfer Protocol

HTTP is used by web applications to communicate over the Internet based on the request-response paradigm [5]. Clients communicate with servers by sending requests that are formed according to the protocol and then the server sends a response back to the client. When a client sends a request over the HTTP, it is usually done using one of the following common methods:

- \textit{GET}: used when requesting a resource over HTTP. The resource can be almost anything including web pages, images and video files;

- \textit{HEAD}: like the \textit{GET} method, but it only returns the \textit{head} part of the response, excluding the \textit{body} part. The \textit{head} is the part containing the response headers;

- \textit{POST}: used when uploading data, images or video to a server.

The HTTP is the foundation of data communication for the World Wide Web (WWW) and, initially, had three basic features that made it a simple, but powerful protocol: connection-based, media independent and stateless. In the newer versions, HTTP provides persistent connections through a keep-alive-mechanism. With this mechanism, a connection can be reused for multiple requests.

2.2.2 Remote Procedure Call

RPC is an Interprocess Communication (IPC) protocol that allows the exchange of data and invocation of certain features that are in different processes. This protocol uses the client-server model where the client is the requesting program and the server is the SP program. A RPC is a synchronous operation requiring the client to be suspended until the results of the requests are returned [6]. However, the use of lightweight processes or threads that share the same address space allows for multiple RPCs to be performed concurrently.

\textsuperscript{2}https://cakephp.org , last accessed July 3rd, 2017
When program statements that use RPC are compiled into an executable program, a stub is included in the compiled code that receives the requests and forwards them to a server runtime program. The client runtime program has the knowledge of how to address the remote computer and server application, sending the requests across the network. Similarly, the server includes a runtime program and a stub that interfaces with the remote procedure itself.

This protocol is normally used in Asynchronous JavaScript and XML (AJAX) applications, but uses JavaScript Object Notation (JSON) or eXtensible Markup Language (XML) to encode its calls and HTTP as a transfer mechanism [7].

2.2.3 WebSocket

The WebSocket protocol represents a long-awaited evolution in client-server web technology. This protocol allows a long-held single Transmission Control Protocol (TCP) socket connection to be established between client and server. It uses bi-directional and full duplex communication channels, which enable messages to be instantly distributed with little overhead resulting in a very low latency connection [8].

WebSocket is designed to be implemented in web browsers and web servers, allowing for more interaction between them and facilitating the real-time data transfer from and to the server with fewer delays. Using this protocol can bring some advantages, such as less network resources consumption and no problem with connection limitations since a single TCP socket connection is used.

However, there are still some concerns about this protocol because many libraries and technologies understand HTTP, but might not be WebSocket ready.

2.2.4 WebRTC

The WebRTC standard is an HTML5 technology driven by web companies, commonly named OTT. It aims at offering native browser tools the ability to insert real-time media stream exchange services easily into the web applications. This means that the user does not have to download, install and manually configure an application or to use some proprietary plug-in in the browser [9].

This standard provides several JavaScript Application Programming Interfaces (APIs) (e.g. RTCPeerConnection and RTCDataChannel), allowing the data streaming between browsers. However, there are two very important aspects that should be considered. First, it is also necessary to provide a mechanism to coordinate communication and to send control messages, a process known as signaling. Suitable signaling methods and protocols should be chosen depending on the use case [10]. Second, WebRTC is designed to work P2P, so users can connect by the most direct route possible. Nevertheless, WebRTC is built to cope with real-world networking, where client applications may need to traverse Network Address Translation (NAT) gateways and firewalls. As such, P2P networking needs fallbacks in case direct connection fails. Usually, to overcome this second aspect, the Interactive Connectivity Establishment (ICE) protocol is used, which allows the establishment of multimedia sessions based on the request-response model. It uses Session Traversal Utilities for NAT (STUN) servers to get the Internet Protocol (IP) address of the computer and Traversal Using Relays around NAT (TURN) servers to function as relay
servers in case P2P communication fails. ICE protocol works with high efficiency, even in very complex topologies prone to failures [11, 12].

In terms of security in WebRTC, the data is normally encapsulated and encrypted through the Datagram Transport Layer Security (DTLS) protocol, which provides data integrity and confidentiality [13]. After that, the encapsulated and encrypted data should be transmitted using a Secure Socket Layer (SSL) connection [14].

The WebRTC standard also provides an authentication mechanism using a web-based Identity Provider (IdP). The idea is that the entity sending an offer or answer acts as the Authenticating Party (AP), getting an identity assertion from the IdP and attaching it to the session description. The consumer of the session description acts as the Relying Party (RP) and verifies the assertion. The interaction with the IdP is designed to decouple the browser from any IdP, supporting several identity protocols.

### 2.3 Security in Web Development

Another very important aspect that should be considered is security. Usually, developers need to write code that fulfills customer functional requirements, and need to be it fast. Somewhere, way down at the bottom of the list of requirements, behind, fast, cheap and flexible is “secure”. That is, until something goes wrong, until the system built is compromised, then suddenly security is the most important thing.

Web application development considers many security considerations such as data entry error checking through forms, filtering output, cryptography, secure communications and sandboxing mechanisms.

#### 2.3.1 Sandboxing Mechanism

Sandbox is an encapsulation mechanism used to impose a security policy on software components [15]. Usually, this mechanism is used to execute untested or untrusted programs or code that may contain a virus or another malicious code, possibly from unverified or untrusted third-parties, suppliers, users or websites without affecting the application in which it runs.

A sandbox typically provides a tightly controlled set of resources for guest programs to run in, such as space on disk and memory. On the other hand, network access, the ability to inspect the host system or read from input devices are usually disallowed or heavily restricted. The sandbox implementation should provide certain mechanisms to validate the untrusted programs through security rules, validation of data and code checkers.

Implementing this mechanism on the applications could bring some advantages, such as no need to develop other security policies elsewhere. Despite that, there are some concerns like complexity, most notably about how to get desired changes to be preserved outside of the sandbox.

#### 2.3.2 Cryptography

Cryptography has a huge impact in the security of web applications, namely in terms of information security. This mechanism can provide data confidentiality, data integrity, non-repudiation and secure
authentication and communication.

However, bad choices by the developer may lead to vulnerabilities that may be exploited by malicious users with the aim of obtaining access to sensitive application information, such as authentication credentials.

In order to avoid those bad choices, the developer should perform some verifications to the application. First, the developer should check if there exits data or exchanges of data, which should be encrypted, but are not; and second, should check for wrong, weak or predictable algorithms usage, depending on the context. There are several cryptographic algorithms, some better for secure communications (e.g. Transport Layer Security (TLS) protocol [16]) and some better for authentication credentials storage (e.g. Bcrypt [17]). Considering this, the programmer should choose the most appropriate algorithms.

2.3.3 Secure Communication

Communication has a very important role in web applications, but it may be subject to numerous vulnerabilities. As such, the developer must use secure protocols, enabling the creation of secure communication channels. A secure communication occurs when two entities are communicating and a third-party cannot listen in. In addition, a secure communication should provide authenticity, confidentiality, identification mechanisms and non-repudiation.

Secure communications over HTTP are mainly achieved through the SSL/TLS protocols, i.e. using the Hypertext Transfer Protocol Secure (HTTPS). These two protocols provide four main features and protections:

- Server authentication that is based on the server certificate. Clients can detect whether an unauthorized entity is trying to impersonate a legitimate web server;
- Mutual authentication between servers and clients based on both clients and servers having trusted certificates;
- Confidential web communications that use an encrypted and secure channel;
- Data integrity that is based on Hash Message Authentication Codes (HMAC) [18].

As mentioned above, a mutual authentication protocol can be used to ensure that the communication channel is secure and trustworthy. This protocol, also provides two-way authentication, allowing two parties to authenticate each other at the same time, ensuring that they are doing business exclusively with legitimate entities and servers. With mutual authentication, a connection can occur only when the client trusts the server certificate and the server trusts the client certificate.

2.3.4 Validation

Web applications are notorious for taking practically any type of input, assuming that it is valid, and processing it further. Not validating the input is one of the greatest mistakes that web application devel-
opers can make. Among the classes of vulnerabilities exhibited by web applications, Cross-Site Scripting (XSS) and Structured Query Language (SQL) injection remain among the most serious threats to web application security that benefit from the lack of validation mechanisms in the applications [19].

There are two main types of validation techniques that a developer should implement to protect the applications against those vulnerabilities: the validation performed over the input data, such as data types validation and range and constraint validation; and the validation performed over the output data, automatically applied to data computed from untrusted data.

The developer should keep this aspect in mind since a simple and an effective validation mechanisms can be enough, in many cases, to protect applications from several typical web attacks.

2.3.5 Typical Web Vulnerabilities

Attackers have an ever-growing list of vulnerabilities to exploit and to maliciously gain access to web applications, networks and servers. New vulnerabilities are being discovered all the time by security researchers, attackers and even by users. Each time changes are made at any level of the infrastructure, there is the potential for new vulnerabilities to appear.

The Open Web Application Security Project (OWASP) releases from time to time a list of the top 10 critical web application security flaws. The actual list includes vulnerabilities such as Injection, Broken Authentication and Session management, XSS and Cross-Site Request Forgery (CSRF) [20].

Developers must seriously take these vulnerabilities into account when programming their applications. As already mentioned, using simple mechanisms, cryptography, secure communication and authentication can avoid such vulnerabilities with a minimal effort.

2.4 Web based Identification and Authentication

The identification and authentication of users is also another very important aspect in web application development. Web applications, in general, use HTTP as the application-layer protocol, which is perhaps the most popular application protocol used on the Internet [21].

This protocol, as already mentioned, is a stateless protocol, in other words the current request does not depend on the previous requests. For this reason, the web server does not retain any information or status about each communication partner across multiple requests.

However, for most web applications it is almost mandatory to know who connects to the server, for tracking an application usage, the general interaction patterns of users and to deliver the dynamic content requested. To solve this problem, the applications need to implement mechanisms that provide two important features: the association of a statement to the communication partner (for identification), that can be name, e-mail address or many other things; and that proves the identity that the user says to be his (for authentication).
2.4.1 HTTP based Authentication

The HTTP provides a few authentication schemes to handle user authentication. For now, we concentrate only on two authentication schemes that differ on the security level (both are highly insecure): basic authentication and digest authentication [22].

**Basic Authentication:** In the basic authentication scheme, all users must authenticate themselves with a username and password. As represented in Figure 2.1, initially, the server denies the client request with a WWW-Authenticate response header and a 401 unauthorized status code. On seeing this header, the browser displays a login dialog, prompting for a username and password. This information is sent in a base-64 encoded format in the Authentication request header. After that, the server can validate the request and allow access if the credentials are valid.

This scheme has two main problems: first, it does not have any mechanism available for the server to cause the browser to discard the stored credentials for the user (logout), and second, the username and password travel in effective clear-text, which is extremely vulnerable to attacks.

To provide transport security and to overcome the second problem, the system designer needs to use secure protocols, such as HTTPS, to provide confidentiality and integrity in transit.

![Figure 2.1: HTTP Basic authentication scheme using HTTPS](image)

**Digest Authentication:** The digest authentication scheme, represented on Figure 2.2, is similar and uses the same handshake technique with the WWW-Authenticate and Authentication headers. But this scheme uses a more secure hashing function to encrypt the username and password, commonly with Message Digest 5 (MD5) [23].

This scheme is the least popular between the two because of its complexity, however, this one can provide integrity protection and mutual authentication.

![Figure 2.2: HTTP Digest authentication scheme](image)
2.4.2 Cookies

The goal of identification is to adapt the response to provide a personalized experience and to achieve this, the server must know who a user is, making the web application stateful. There are a few different ways a server can collect this information such as client-IP, request headers and the most popular and non-intrusive: cookies.

Cookies allow the server to attach arbitrary information to outgoing responses via the *Set-Cookie* response header [24]. This arbitrary information can be a lot of things, such as a user identifier or a database key, whatever the server needs, so it can continue where it left off. Under normal circumstances, a cooperating client returns the cookie information, each time it makes a new request to the same server.

This approach has a variety of uses, among which is storing the login information for applications that provide personalized authentication, so the user does not have to keep entering his credentials. Even if web applications use another more complex authentication mechanism, it still uses cookies, after the first authentication to manage each new session.

Most web applications use cookies as the only identifiers for the user sessions, making the applications vulnerable to attacks, such as cookie session hijacking, cookie poisoning and XSS [25]. Although most vulnerabilities could be solved with the use of HTTPS, developers should be careful when using this mechanism, choosing implementations that provide greater security.

2.4.3 Local Password based Authentication

Rather than relying on authentication at the protocol level, web based applications can use embedded code on the web pages themselves. The developers use the HTML capabilities, especially through forms to request the authentication credentials (username and password). Web applications that use this type of authentication need to implement their own protection against the classic protocol attacks and build suitable secure storage of the encrypted password repository.

Local password based authentication requires the developer to implement an authentication protocol considering the same problems that HTTP Digest authentication was created to deal with. Specifically, the developer should remember that forms submitted using HTTP capabilities will send the credentials in effective clear-text, unless HTTPS is used.

2.4.4 Digital Certificates

A digital certificate provides identifying information, allowing a person, computer or organization to exchange information securely over the Internet using the Public Key Infrastructure (PKI). The Certificate Authority (CA) acts as the trusted intermediary that issues digital certificates, associating a given public key (and consequently the respective private key) with a given identity, enabling the ownership of the certificate to be verified unambiguously.

On the client side, the certificate together with the private key can be stored by the operating system or by the browser, in a file or in a separate physical device. Usually, the private key is password-
protected. This mechanism is also well supported by browsers, allowing the user to select and apply the certificate if the web application allows such way of authentication [26].

Using certificates for authentication is a much more reliable way of authentication by means of passwords, and provides important advantages to the applications, e.g. there is no need to transmit a secret and allow for non-repudiation. However, there are some difficulties that the developer must handle too, e.g. the complexity of the infrastructure and the certificate status update and reporting.

2.4.5 Token based Authentication

Authentication based on tokens is most often applied in the creation of distributed systems with Single Sign-On (SSO) [27], where one application, SP or RP, delegates the function of authentication of users to other application, IdP or AP. In this approach, the IdP provides the authentic information about the user in the form of a token and SP applications use this token for identification, authentication and authorization of users.

There are some standards that define the protocol of interaction between applications for this type of authentication. Within the context of this work, the most relevant standards are: OAuth and OpenID.

**OAuth standard:** OAuth is an open standard for authorization, commonly used as a way for Internet users to authorize applications to access their information on other applications, but without giving them the passwords. This mechanism is used (e.g. by Google, Facebook, Microsoft and Twitter) to enable users to share information about their accounts with third-party applications [28].

Generally, OAuth provides a secure delegated access to server resources on behalf of a resource owner and its current version is OAuth 2.0 [29]. This standard specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials. Designed specifically to work with HTTP, OAuth essentially allows access tokens to be issued, with limited privileges and for a limited time, to third-party clients by an authorization server, with the approval of the resource owner. Firstly, the client request authorization to the resource owner. Then, the resource owner grants permission to client for access to resource in the form of a grant. This grant can be, e.g. an authorization code that is generated after the successful authentication of the client on the authorization server. Thirdly, the client addresses the authorization server and receives an access token to the resource in exchange for the grant. The client then uses the access token to access the protected resources hosted by the resource server. This component interaction is represented on Figure 2.3.

Using this standard could result in many advantages to the web applications and to the users. This standard makes the authentication not only easier, but incredibly time saving in the long run. Another important aspect is the security. In the current version, all OAuth data transfers must take place over TLS to ensure that the most trusted cryptography industry protocols are being used to keep data as safe as possible.

On the other hand, it has some limitations. An important one is the lack of anonymity, because this standard does not provide some alternative feature that allows users to use the web applications anony-
The OpenID standard is an open standard and decentralized authentication protocol that allows users to be authenticated by co-operating sites using a third-party service, eliminating the need for web applications to provide their own ad hoc login systems, and allowing users to log in to multiple unrelated applications without having a separate identity and password [30].

Its current version is OpenID Connect, which combines the OpenID and OAuth standards. This version allows clients to verify the identity of an end-user based on the authentication performed by an authorization server as well as to obtain basic profile information about the end-user.

Users create accounts by selecting an OpenID Identity Provider (OP), and then use those accounts to sign onto any application which accepts OpenID authentication. The OpenID standard provides a framework for the communication that must take place between the OP and the OpenID acceptor or RP. An extension to the standard (the OpenID Attribute Exchange [31]) facilitates the transfer of user attributes, such as name and gender from the OP to the RP. This component interaction is depicted on Figure 2.4.

The OpenID Connect provides a Representational State Transfer (REST) API, using JSON as the data format. Using this standard can also result in many advantages to the web applications and to the users. This standard not only allows the verification of the user identity, but also gain limited access to the end-users' information. Another aspect is that it empowers the user with the choice of the identity because it is very easy to maintain multiple identities.

However, there are still some concerns regarding this standard and some limitations. An important concern is security, since OpenID is vulnerable to phishing attacks [32] due to the reliance on a second application for sign-in and the solution for this could be very complex. Additionally, this standard has the same limitation of the OAuth 2.0, the lack of anonymity.
2.5 Chapter Summary

In the previous Sections was presented an overview about web application development, focusing on important aspects that every developer needs to consider while developing this type of applications. We have described several protocols and techniques used nowadays in different aspects of the web application development, such as security and authentication of users.

The aim of this research is to highlight the advantages and drawbacks of each one, making it easier to select the most suitable protocols and techniques for the development of web applications.
Chapter 3

Related Work

This Chapter addresses the state of the art of frameworks for web application development. First, we provide an overview on some considerations that a programmer should have when choosing a framework for web application development. Secondly, a reflection on how to evaluate the available frameworks is also provided. Then, we present a better insight about the reTHINK framework, its architecture and an example of a complete operation with this framework. To conclude, we provide a comparative analysis between several types of frameworks, regarding key aspects such as programming language, communication protocol, security and authentication mechanisms.

3.1 Framework Selection

In this current day and age, pushing out a finished and polished application well before a competitor is key for success. Coding everything from scratch, excluding even the mundane things, can be extremely time-consuming and makes the developer waste time reinventing the wheel, time that can rather be spent developing and implementing new features or tightening up the code base.

This is where web development frameworks come in. A framework in the context of software development is a set of prewritten code or libraries that provide functionality common to a whole class of applications. A framework can be seen as a base or a skeleton to build upon, allowing to speed up development by not having to rewrite features and structures that are commonly used in most web applications. Instead of re-inventing the wheel repeatedly, the developer has many wheels (functions/features) that are already tested and working properly.

Consequently, choosing a framework for application development is a very important decision. The developer should look for a framework that is appropriate for the targeted application, that uses the desired programming language and that is based on a suitable software architectural pattern to help him organize the application development (e.g. Model-View-Controller (MVC) [33]). In addition, it should cover all the usual services needed on an application, including identification, authentication, communication, security and database mapping.
3.2 Framework Evaluation

A framework usually provides a long list of features, which the developer can use, to facilitate in the development process. However, there is a lack of a unified list of commonly used features, which indicates if a framework has the ability to aid the developer with some task. This can lead to the inability to make a proper choice whether a framework fulfills ones needs or not.

Thus, there is a need to level the playing field for the frameworks by introducing a list of features that frameworks should offer. The features, which should be contained in the list, need to cover commonly used functionality in web applications and more specifically, functionality that facilitate rapid development doing so by lowering the programming effort. Other aspects should also be considered, because they have a greater involvement in the programming effort, such as the available documentation, learning curve, portability, usability and existing community [34, 35].

With this in mind, a methodology to evaluate frameworks for web application development should be based on several criteria, not only related with the features and the application requirements, but also with the ease of use and learning. This methodology should always take into account that a framework must always speed up the development process and decrease the programming effort.

3.3 The reTHINK framework

The reTHINK project describes a new framework for web application development, especially for those with communications capabilities, providing real-time communication over the Internet in a secured way and possibly with enhanced Quality of Service (QoS). This framework incorporates two software concepts (the ProtoFly and Hyperty) and follows the Microservice software architectural pattern.

The Hyperty is a new web service that can be described as a module of software dynamically deployed in web runtime environments on end-user devices, through simple, but sophisticated identity management techniques. These Hyperties are maintained by the SPs and their Catalogue services helps the user to choose and download the most appropriate Hyperty instance. The Hyperty deployed on the runtime of the users’ device represents a live user who is available for incoming communications from the particular SP. The communication between Hyperties is based on the ProtoFly concept, which allows the use of standard network protocols without the need to modify them or to create new ones. The ProtoFly pulls code on-demand to dynamically select, load and instantiate the most appropriate protocol stack (e.g. JavaScript file) during runtime. The implementation of this protocol stack is called Protocol Stub (ProtoStub). This concept enables protocols to be selected at runtime and not at design time, enabling protocol interoperability among distributed services, promoting loosely coupled service architectures and optimizing resources spent by avoiding the need to have protocol gateways in services middleware.

The communication between different Hyperties can be supported by P2P connections or via a Message Node. In both options, a ProtoStub implementing the most appropriate network protocol stack is used, e.g. JSON over WebSockets or a REST API Client are good options for ProtoStubs used to
interface with a Message Node, while the WebRTC DataChannel API is a good option for a ProtoStub used to directly interface with another Hyperty.

In terms of identification and authentication, Hyperties extend the WebRTC identity model [36], where identity tokens are generated and inserted, by the runtime, into intercepted messages sent by Hyperties and validated by the recipient Hyperty’s runtime. The Identity Module (IDM) is the core runtime component responsible for handling the user identity and to associate it with the Hyperty instances. The identity in the reTHINK framework is not fixed to a unique identity source, so the user can choose the preferred IdP for authentication.

To ensure secure communications, reTHINK uses HTTPS and the Mutual authentication protocol. Every time a user starts a communication with another user, the process of mutual authentication is initiated by the IDM using also the OpenID Connect protocol. Mutual authentication is not only useful for the authentication of users, it is also essential for the exchange of the symmetric keys used in the establishment of secure communications [37]. In addition, reTHINK implements sandboxing mechanisms on the Hyperty runtime that ensures the correct isolation of the client side JavaScript code (i.e., Hyperties, ProtoStubs and Applications). Communication outside the sandbox is possible through well-defined channels.

One additional feature provided by the reTHINK framework is the discovery service through a Directory component. This service provides means for users to find other users to initiate the communication with, without any need to be a subscriber of a certain communication service.

Although the reTHINK framework appears to be a very complete framework, it lacks in terms of features related with the client side, such as integration with several template engines, features to deal with the user interface and built-in integration with front-end frameworks. In addition, there are some concerns associated with the size of the framework (6.8 MB for the Runtime only) that will affect the deployment time of the developed applications.

3.3.1 Architecture

To provide a better insight to the reTHINK project architecture, its five main components are herein described, namely: Global Registry, Discovery Service, SP, IdP and the user device. Figure 3.1 depicts the interconnection between these five components and their details, also illustrating their internal components.

The user device is the component responsible for running the reTHINK client side application deployed from the Web Server. This application is composed of one or more Hyperty instances and the Runtime Core, both downloaded from the SP Catalogue. The Hyperty can be obtained from the SP whenever a user decides and it can be deployed in a wide variety of devices ranging from smartphones, laptops and cloud infrastructures. The reTHINK framework running in the user device has no limits regarding the number of Hyperties running at the same time. The Runtime Core is the component responsible for ensuring the correct functioning of the reTHINK application as well as managing the deployed Hyperties. This component has the responsibility to install the Hyperties, ensure the security
aspects of the application, manage the identities obtained from the IdPs and handle all the messages entering and leaving the application. To fulfill all these services, the Runtime Core has integrated some additional components, such as: Message Bus, Policy Engine, IDM, Graph Connector and Discovery. All these components are deployed once and executed in the background, avoiding the need to be constantly downloading the Runtime Core unless there is a more recent version.

A SP is comprised by several internal components such as Domain Registry, Message Node and Catalogue. The Domain Registry has the responsibility to register publicly the Hyperties instances upon their installation in the user device. A list of registered Hyperty instances for a specific user can be retrieved through a lookup function. The Message Node is the component responsible for intercepting and forwarding messages to be exchanged between users. These messages are usually signaling messages used to initiate a call. The Catalogue is a repository provided by the SP that hosts the web oriented software to be used by user devices, e.g. the Hyperty code. The user device can communicate with the SP through the installation of a ProtoStub provided by the SP catalogue service.

The long and complex strings derived from the Hyperty instances Uniform Resource Locators (URLs) can be difficult for humans to memorize and use as a contact number. To overcome this, Hyperties registered in the user device need to have a user identity associated to it. With this association becoming publicly registered, it is possible to discover the user Hyperty instance by searching for a known user identity, e.g. Google email address. In addition, reTHINK users have access to the Discovery Service, where they can create profiles. With this service, users can search others using their profile information such as favorite sport, football club and favorite movies.

In reTHINK, users (entities) are identified by globally unique, domain-agnostic identifier called Global User Identifier (GUID) that remain stable and unchanged throughout the entire lifespan of a user account and it can be used to address and identify users and devices [38]. Whenever a user uses one of his identities to associate it with an Hyperty, this identity becomes associated to his GUID independently from the IdP used. With this, the GUID can have multiples identities linked to it, ensuring that all those identities belong to the same user. The public service called Global Registry stores the GUIDs and
the associated identities used to register Hyperties. This component is deployed in a Distributed Hash Table (DHT) and it has the role of resolving a given user identifier to the SP domain that is registered.

3.3.2 Usage Example

To explain how the components and identifiers operate in reTHINK framework, we will demonstrate a use case where Alice initializes the reTHINK application and then tries to contact Bob, who is already running some reTHINK application. Figure 3.2 demonstrates the flow of operations that need to be executed to achieve the communication between them.

![Figure 3.2: reTHINK framework usage example](image)

When Alice starts the reTHINK application deployed from the Web Server, the Runtime will be installed and the selected Hyperties will be loaded into the application. To successfully deploy an Hyperty, authentication is mandatory. So, Alice is prompted to obtain an identity from the preferred IdP that will verify Alice’s credentials and retrieve the idAssertion (step 1 of Figure 3.2). After the Runtime Core receives the identity, it starts downloading and installing the Hyperty from the Catalogue of SP A (step 2 of Figure 3.2). Once the installation is done, an Hyperty instance URL is generated and the Runtime Core associates Alice’s identity with the Hyperty instance URL to register it in the Domain Registry of the SP A (step 3 of Figure 3.2).

To establish a communication channel, Alice needs to discover Bob’s Hyperties instances (step 4 to 7 of Figure 3.2). Firstly, Alice tries to discover his profile using some profile information that she knows, e.g. email, favorite football club. The Runtime Core uses the Discovery Service to search for Bob’s
profile to obtain his GUID. Assuming that Bob has a profile, a query to the Global Registry is issued to resolve Bob’s GUID into his IDs and SP domains. With Bob’s IDs, it is now possible to query the Domain Registry of the Bob’s SP that will reply to Alice’s device with Bob’s Hyperty instance URL, for which he can be contacted.

Steps 8 to 11 represents the call initiation between Alice and Bob. The call initiated by Alice contains her idAssertion that will be validated by Bob’s Runtime Core through the Alice’s IdP. After validation, Bob’s Runtime Core initiates a mutual authentication process to authenticate themselves mutually and to exchange the keys used to establish a secure communication channel.

Most of these steps are transparent to the user. The user is only responsible to login into an IdP to obtain his identity, choose the Hyperty to install and search for another user with a known identity or profile information in order to start a call.

3.4 Full-Stack Frameworks

Frameworks are like jet packs for web application development, allowing to boost performance and extend capabilities. They are not just bundled snippets of code. Frameworks offer features such as models, APIs and other elements to streamline development of dynamic and rich web applications. While some of them offer a more rigid approach to development, others allow a more fluid process. Usually, there are two classifications of frameworks for web application development: full-stack framework and non-full-stack framework.

In this section, we will perform a comparative analysis between full-stack frameworks commonly used in web application development. The most well documented frameworks for several programming languages were chosen and the main goal of this analysis is to highlight the main differences between them, in particular with the reTHINK framework. The results of the performed comparative analysis are represented in Table 3.1.

The following frameworks are all classified as full-stack framework, which means they help the developer with the full development stack from the UI (client side) down to the data store (server side).

3.4.1 Sails.js

Sails.js\(^1\) is probably the most popular MVC framework for creating Node.js applications, providing full compatibility with front-end tools, e.g. template engines. This framework works with two different middleware: the Connect middleware\(^2\) and an additional configurable middleware that needs to be developed from scratch. The main differences between them is that the configurable middleware can be used only for handling HTTP requests, while the Connect middleware is already a wealthy middleware that can be used not only for handling HTTP requests, but also for WebSockets and for the definition of policies to work with any authentication scheme, cookies and access control mechanisms.

\(^1\)http://sailsjs.com, last accessed July 11th, 2017
\(^2\)https://github.com/senchalabs/connect, last accessed July 11th, 2017
In terms of security, Sails.js provides built-in, easily configurable, protection against most known types of web attacks and enforces secure communications. To protect the applications against CSRF and Socket Hijacking, Sails.js implements the Synchronizer Token Pattern [39], forcing all non-GET requests to the server to be accompanied by this special token, identified by either a header or a parameter in the query string or HTTP body. For secure communications, Sails.js uses the Strict Transport Security (STS) protocol [40] that forces the usage of HTTPS instead of HTTP.

With this framework, it is not necessary to program a custom authentication scheme, because it works with an authentication middleware called Passport\(^3\) that provides several web based authentication schemes, such as local password, HTTP (basic and digest) and tokens (OAuth and OpenID).

In addition, Sails.js uses a powerful database mapper called Waterline\(^4\), working as Object Relational Mapper (ORM). This tool dramatically simplifies interaction with one or more databases, providing an abstraction layer on top of the underlying database, allowing the programmer to easily query and manipulate data without writing vendor-specific integration code.

3.4.2 Ruby on Rails

Ruby on Rails\(^5\) is a web application development framework written in the Ruby language. It is designed to make programming web applications easier by making assumptions about what every developer needs to get started. This MVC framework provides, by default, a web server called Puma\(^6\) to handle the HTTP requests. In its current version, it is also possible to communicate over WebSockets using the Action Cable component based on the Publish-Subscribe pattern.

In terms of security, this framework provides a full security guide describing common security problems, e.g. Session Hijacking, XSS and SQL injection, in web applications and how to avoid them in Ruby on Rails. For secure communications, it provides a configuration that forces the use of HTTPS.

Ruby on Rails uses Cookies as the only identifiers for user sessions and provides a very simple HTTP authentication system (basic). For the database mapping, this framework provides an implementation of the Active Record pattern [41], which itself is a description of an ORM system.

One final feature is the packaging system called RubyGems that Ruby on Rails supports. With this system, it is very easy to download, install, and use external Ruby software packages related to authentication schemes, communication and many others.

3.4.3 Laravel

Laravel\(^7\) is a powerful PHP framework, designed for developers who need a simple and elegant toolkit to create full-featured web applications. This MVC framework provides an official, pre-packaged Vagrant box\(^8\) (a simple, elegant way to manage and provision Virtual Machines) called Homestead. With this

\(^3\)http://passportjs.org/, last accessed July 11th, 2017
\(^4\)https://github.com/balderdashy/waterline, last accessed July 11th, 2017
\(^5\)http://rubyonrails.org, last accessed July 11th, 2017
\(^6\)http://puma.io/, last accessed July 11th, 2017
\(^7\)https://laravel.com, last accessed July 11th, 2017
\(^8\)https://www.vagrantup.com/, last accessed July 11th, 2017
box, there is no need to install PHP, a web server or any other server software on the local machine.

The programmer needs to configure a middleware responsible for handling HTTP requests and access control through policies and gates. Laravel works also with WebSockets through a JavaScript library called Laravel Echo and provides a simple templating engine called Blade to handle with frontend capabilities. In terms of security, this framework is probably the most secure on this survey, providing protections against several web attacks, a cryptography component, a hash interface, some validation approaches for incoming data and secure communications. To protect the applications against CSRF and Socket Hijacking, Laravel uses the same mechanism as Sails.js, where the middleware is responsible for validating the token on each request. The cryptography component uses the OpenSSL library to provide AES-256 and AES-128 encryption and the hash interface provides secure hashing for storing user passwords using the Bcrypt algorithm. To ensure secure communications, Laravel uses HTTPS when exchanging sensitive data.

This framework uses cookies as the only identifier for user sessions and provides several authentication schemes, such as local password, HTTP authentication (basic), an authentication interface based on OAuth protocol, and also supports the Passport middleware like Sails.js does.

Another great feature of Laravel is the database mapper called Eloquent, which is an Active Record implementation (ORM).

### 3.4.4 CakePHP

CakePHP\(^9\) is a modern MVC PHP framework. It is intended to make developing, deploying and maintaining applications much easier. This framework provides a simple middleware to handle the HTTP requests.

In terms of security, it provides a library to handle basic security measures such as methods for hashing and data encryption using, e.g. OpenSSL libraries. In addition, it provides three security components: one to protect the applications against some web attacks such as CSRF and Session Hijacking based on the same mechanism as Sails.js, another to force secure communications using the HTTPS protocol and one last component to perform some validations over incoming data.

CakePHP uses cookies as the only identifier for user sessions and provides an authentication component, which has some authentication schemes such as local password and HTTP authentication (basic and digest). For database mapping, this framework provides a hybrid implementation of an ORM based on the Active Record.

### 3.4.5 Django

Django\(^10\) is a high-level Python web framework that encourages rapid development and clean and pragmatic design, also supporting the MVC pattern. This framework provides a built-in middleware to deal with HTTP requests. For identification and authentication of users, it uses cookies and local Password are used.

\(^9\)https://cakephp.org/ , last accessed July 12th, 2017

\(^10\)https://www.djangoproject.com , last accessed July 12th, 2017
In terms of security, Django provides some templates that help the developer protect the applications against typical web attacks such as SQL injection and XSS, through the escaping of specific characters that are particularly dangerous in HTML. In addition, it forces the use of HTTPS to exchange sensitive data and performs some validations to the incoming data.

This framework provides a built-in ORM simplifying the interaction with one or more databases and uses a default template engine developed using the Django template language.

### 3.4.6 Tornado

Tornado\(^\text{11}\) is a MVC Python web framework and asynchronous networking library. By using non-blocking network I/O, Tornado can scale to tens of thousands of open connections, making it ideal for long polling WebSockets, and other applications that require a long-lived connection to each user. This framework provides a client and server side implementations supporting HTTP and WebSockets.

This framework uses cookies as the only identifiers supported for user sessions and provides an authentication module based on the OpenID and OAuth protocols for third-party authentication. In its current version, Tornado supports the HTTPS protocol, ensuring secure communications and provides some security protections against typical web attacks like CSRF.

Taking into account all the information available in Table 3.1, we can conclude that the reTHINK framework should not be classified as a full-stack framework. Although the reTHINK framework provides multiples features in terms of communication, security, identification and authentication of users, it is very limited when we look for example to the client side and data storage. We can even say that the typical views, i.e. to render web pages, are out of scope of the reTHINK framework.

In addition, the use of the Microservice software architectural pattern could be target of some criticism. This software architectural pattern allows structuring an application as a collection of loosely coupled services. The benefit of decomposing an application into different smaller services is that it improves modularity and makes the application easier to understand, develop and test [42]. However, these benefits may not happen as we shall see in later chapters, in which for example test and deployment are more complicated.

\(^{11}\text{http://www.tornadoweb.org , last accessed July 12th, 2017}\)
<table>
<thead>
<tr>
<th>Frameworks</th>
<th>Language</th>
<th>Software Architectural Pattern</th>
<th>Communication</th>
<th>Security</th>
<th>Identification/Authentication</th>
<th>Database Mappers</th>
</tr>
</thead>
<tbody>
<tr>
<td>reTHINK</td>
<td>JavaScript</td>
<td>Microservices</td>
<td>WebRTC; WebSockets</td>
<td>Sandboxing mechanism; Secure Communications (HTTPS, Mutual authentication)</td>
<td>WebRTC Identity; Third-party login (OpenID Connect)</td>
<td>N/A</td>
</tr>
<tr>
<td>Sails.js</td>
<td>JavaScript</td>
<td>MVC</td>
<td>WebSockets</td>
<td>Built-in protections against web attacks; Secure communication (HTTPS)</td>
<td>Cookies; External Middleware (Local Password, OpenID, OAuth)</td>
<td>External ORM</td>
</tr>
<tr>
<td>Ruby on Rails</td>
<td>Ruby</td>
<td>MVC</td>
<td>WebSockets</td>
<td>Built-in protections against web attacks; Secure communication (HTTPS)</td>
<td>Cookies; HTTP Authentication (Basic)</td>
<td>Built-in ORM</td>
</tr>
<tr>
<td>Laravel</td>
<td>PHP</td>
<td>MVC</td>
<td>WebSockets</td>
<td>Cryptography component (OpenSSL, Bcrypt); Built-in protections against web attacks; Secure communication (HTTPS); Validation of data</td>
<td>Cookies; Local Password; HTTP Authentication (Basic); OAuth based interface; External Middleware (Local Password, OpenID, OAuth)</td>
<td>Built-in ORM</td>
</tr>
<tr>
<td>CakePHP</td>
<td>PHP</td>
<td>MVC</td>
<td>N/A</td>
<td>Cryptography library (OpenSSL, Bcrypt, MD5, SHA1, SHA256); Built-in protections against web attacks; Secure communication (HTTPS); Validation of data</td>
<td>Cookies; Local Password; HTTP Authentication (Basic or Digest)</td>
<td>Built-in ORM</td>
</tr>
<tr>
<td>Django</td>
<td>Python</td>
<td>MVC</td>
<td>N/A</td>
<td>Built-in protections against web attacks; Secure communication (HTTPS); Validation of data</td>
<td>Cookies; Local Password</td>
<td>Built-in ORM</td>
</tr>
<tr>
<td>Tornado</td>
<td>Python</td>
<td>MVC</td>
<td>WebSockets</td>
<td>Secure communication (HTTPS)</td>
<td>Cookies; Third-party login (OpenID, OAuth)</td>
<td>N/A</td>
</tr>
</tbody>
</table>
3.5 Non-Full-Stack Frameworks

On the other hand, if the framework does not provide a wide range of features to deal both with client side and server side is considered as non-full-stack framework. In this section, we will perform a comparative analysis between non-full-stack frameworks commonly used in web application development. The most well documented frameworks for several programming languages were chosen and the main goal of this analysis is to highlight the main differences between them, in particular with the reTHINK framework. The results of the performed comparative analysis are represented in Table 3.2.

3.5.1 Express

Express\textsuperscript{12} is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web applications, facilitating the rapid development of Node based web applications. This framework is very limited in terms of built-in features, although to add new features it is just a matter of loading an appropriate Node.js driver into the application.

Express uses cookies as the only identifiers for user sessions and it is compatible with a few template engines, such as Pug\textsuperscript{13} and Mustache\textsuperscript{14}. Regarding security aspects, this framework does not have any built-in mechanism to protect the application against typical web attacks. However, it provides a development guide to help the programmer in secure the application. This guide demonstrates how to ensure secure communications with HTTPS protocol, how to use cookies securely and provide some advices to avoid vulnerabilities that could be exploited by a malicious user.

3.5.2 Spark Java

The Spark Java\textsuperscript{15} framework was built around Java 8 lambda function philosophy, making a typical Spark application a lot less verbose than most of the applications written in other Java web frameworks. At less than 1 megabyte in size, Spark Java is an agile and slimmed down micro framework that can be used for quickly and easily developing applications, without the need for a heavy (XML) configuration.

This framework is mainly used for creating REST APIs, using by default an embedded Jetty\textsuperscript{16} web server that supports HTTP and WebSockets. Since Spark Java uses HTTP, this means that the server does not save any state or info related to the users. To solve this problem, this framework uses cookies as the only identifiers for user sessions. In terms of security, this framework is very limited, although secure communications are ensured using the HTTPS protocol.

Spark Java framework has a very limited number of useful features, not providing, for example any database mapper or authentication scheme. With this in mind, this framework is not very well suited for large applications.

3.5.3 Zend

Zend framework\(^{17}\) is an open source framework for developing web applications and services with PHP. The component structure of this framework is somewhat unique, where each component is designed with few dependencies on other components. This loosely coupled architecture allows developers to use components individually.

The wide range of components in the standard library form a powerful and extensible web application framework when combined. Zend framework offers a robust, high performance MVC implementation, a database abstraction that is simple to use and a forms component that implements HTML form rendering, validation and filtering so that developers can consolidate these operations using one easy to use, object oriented interface.

Zend framework provides other components to deal with user authentication and authorization using the HTTP protocol and several libraries that support a few cryptographic tools.

<table>
<thead>
<tr>
<th>Frameworks</th>
<th>Language</th>
<th>Main Focus</th>
<th>Architecture</th>
<th>Built-in Features</th>
<th>Template Engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>reTHINK</td>
<td>Javascript</td>
<td>Protocol independent communication applications</td>
<td>Loosely coupled</td>
<td>Multiple</td>
<td>N/A</td>
</tr>
<tr>
<td>Express</td>
<td>Node.js</td>
<td>Node based web applications</td>
<td>Loosely coupled</td>
<td>Limited</td>
<td>Multiple</td>
</tr>
<tr>
<td>Spark Java</td>
<td>Java</td>
<td>REST APIs</td>
<td>High coupled</td>
<td>Limited</td>
<td>Multiple</td>
</tr>
<tr>
<td>Zend</td>
<td>PHP</td>
<td>General web applications</td>
<td>Loosely coupled</td>
<td>Limited</td>
<td>Limited</td>
</tr>
</tbody>
</table>

The reTHINK framework should be considered as non-full-stack framework. However, it is much more powerful in terms of built-in features than a typical non-full-stack framework, as we can see in this comparative analysis. The reTHINK framework provides built-in features for several aspects such as authentication of users and communication. To provide the same features as reTHINK, typically non-full-stack frameworks need to integrate external components that will provide those features, e.g. WebSocket module.

3.6 WebRTC Frameworks

Some non-full-stack frameworks were built to cope mainly with some technologies or development phases such as with the WebRTC and UI. In this section, we will perform a comparative analysis between non-full-stack frameworks that handle chiefly with WebRTC technology.

\(^{17}\)https://framework.zend.com/, last accessed July 13th, 2017
To deploy a WebRTC application, the developer must deal with the NAT issues at the very least. At the end of the day, this means that deploying more than just web servers and deal with things such as geolocation of target users and other headaches involved with low latency real-time communication. In addition, the required signaling mechanisms are out of the scope of WebRTC and it is left for the developers to decide.

There are already several frameworks that take care of the signaling aspects of WebRTC, which use WebSockets and provide proprietary means to communicate across browsers. The most well documented frameworks were chosen and the main goal of this analysis is to highlight the main differences between them, in particular with the reTHINK framework. The results of the performed comparative analysis are represented in Table 3.3.

3.6.1 PeerJS

PeerJS is a JavaScript framework that simplifies the WebRTC P2P data, video and audio calls to build simple web applications with WebRTC capabilities. With this framework, the identification of peers can be made in two ways: a peer can choose a unique randomly generated ID (string) or have a server generate one for him. PeerJS provides an open source implementation of a web server (PeerJS server) written in Node.js that acts as a connection broker and handles signaling. The ID for identification is passed through WebSockets.

To establish a connection between two peers, one of them must know the ID of the other. For communication, the WebRTC APIs are used to associate data channel to the peers (RTCPeerConnection) and to create interfaces for bidirectional P2P transfers of arbitrary data (RTCDataChannel). This framework was designed to handle only WebRTC, so the programmers should use APIs to handle with other aspects such as: security, database mapping and authentication.

3.6.2 EasyRTC

EasyRTC is a JavaScript framework build on top of WebRTC, supporting the transfer of audio, video and data on a P2P basis putting very little load on supporting servers. This framework consists of a client side JavaScript library and a backend server build on top of Node.js.

The server functionalities are similar to the PeerJS server, where a peer is identified with a unique randomly generated ID (string). However, EasyRTC server provides an additional feature that enables the creation of room levels inside of the same application. These room levels can be viewed as chat rooms with several peers connected. Clients can connect to one or more rooms and see all other connected clients. The server can restrict who gains access to rooms, not allowing the unlimited connection. EasyRTC allows the management of configuration options at the server, application and room level, making it easier to debug in the development phase.

18http://peerjs.com , last accessed July 13th, 2017
19https://github.com/peers/peerjs-server , last accessed July 13th, 2017
20https://easyrtc.com/ , last accessed July 14th, 2017
To start a call between two peers, one of them must know the ID of the other. Then, for the communication, the WebRTC APIs are used for bidirectional P2P transfers of arbitrary data. This framework was built to deal specifically with the WebRTC technology, so the developer should integrate it with other APIs to handle other requirements.

### 3.6.3 SimpleWebRTC

SimpleWebRTC\(^1\) is a JavaScript WebRTC framework that wraps everything the developer needs to build applications with WebRTC capabilities. This framework is constantly updated, which is good because WebRTC is a technology so recent that is updated in a monthly basis. PeerJS and EasyRTC suffer less updates, allowing the appearance of bugs and limitations.

This framework does not provide a server to deal with signaling. However, if the developer does not want to implement it, the creators of SimpleWebRTC suggest the use of a Node.js server called SignalMaster\(^2\) to deal with signaling and the connection of clients. Although this server seems promising, the documentation is very limited.

SimpleWebRTC provides a series of additional APIs to increase the functionality of the application such as an API to detect if someone is speaking or not and an API that provides a simple audio input volume controller.

<table>
<thead>
<tr>
<th>Frameworks</th>
<th>Language</th>
<th>Signaling Handling</th>
<th>Normalizes WebRTC APIs</th>
<th>Peer Identification</th>
<th>Optional Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>reTHINK</td>
<td>Javascript</td>
<td>Yes</td>
<td>No</td>
<td>WebRTC identity model</td>
<td>No</td>
</tr>
<tr>
<td>PeerJS</td>
<td>Javascript</td>
<td>Yes</td>
<td>Yes</td>
<td>Unique Randomly ID</td>
<td>Yes</td>
</tr>
<tr>
<td>EasyRTC</td>
<td>Javascript</td>
<td>Yes</td>
<td>Yes</td>
<td>Unique Randomly ID</td>
<td>Yes</td>
</tr>
<tr>
<td>SimpleWebRTC</td>
<td>Javascript</td>
<td>No</td>
<td>Yes</td>
<td>Unique Randomly ID</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The aim of this comparative analysis is to highlight the main differences between the reTHINK framework and what is used nowadays by frameworks in relation to the WebRTC technology. Taking into account all the information available in Table 3.3, we can conclude that the main differences are the identification of peers and how each handles the signaling process.

### 3.7 UI Frameworks

The UI frameworks, also called front-end frameworks, are packages containing pre-written, standardized code in files and folders. These frameworks give developers a base to build on while still allowing

---

\(^1\)https://simplewebrtc.com/, last accessed July 14th, 2017

\(^2\)https://github.com/andyet/signalmaster, last accessed July 14th, 2017
flexibility with the final design. Usually, UI frameworks contain a grid which makes it simple to organize
the design elements, defined font styles and sizing, and pre-built website components such as side
panels, buttons and navigation bars.

The right UI framework simplifies, streamlines and speeds up the application design and develop-
ment process while still giving the developer the flexibility and features that he needs to produce
exceptional results. However, choose the right one when so many others exist can be tricky. The devel-
oper should look for the pros and cons of the most popular front-end frameworks, in particular with the
following features:

- Responsive design: any developed application should render properly across all devices;
- CSS preprocessors: preprocessors help to achieve writing reusable, maintainable and extensible
codes in CSS. If the developer wants to use one, he needs to ensure that the chosen UI framework
supports it;
- Appearance: the developer should choose a framework that allows to achieve the appearance that
he want with as little effort as possible;
- Prototypes: the ideal UI framework allows the developer to quickly produce wireframes and proto-
types, speeding up the overall design and development process.

In this section, we will do a comparative analysis between UI frameworks. The most well documented
and popular frameworks were chosen. The main goal of this analysis is to highlight the main differences
between them, in particular with the reTHINK framework. The results of the performed comparative
analysis are represented in Table 3.4.

3.7.1 Bootstrap

Bootstrap is the most popular UI framework for developing responsive, mobile first projects on the web.
This framework makes front-end web development faster and easier. It is made for developers of all skill
levels, devices of all shapes and projects of all sizes.

Bootstrap includes CSS, HTML and JavaScript components. It adheres to responsive web design
standards, allowing the programmer to develop responsive sites of all complexities and sizes. This
framework supports the most popular CSS preprocessors to quickly get started with precompiled CSS
or build on the source.

With Bootstrap, the developer has access to an extensive documentation with good quality for com-
mon HTML elements, dozens of custom HTML and CSS components and awesome jQuery plug-ins.
However, the huge amount of styles and components provided can be sometimes a bit messy and
confusing for developers.

http://getbootstrap.com/ , last accessed July 17th, 2017
3.7.2 Semantic UI

Semantic UI\(^{24}\) is a development framework that helps create beautiful, responsive layouts using human-friendly HTML. This framework treats words and classes as exchangeable concepts. Classes use syntax from natural languages such as noun/modifier relationships, word order and plurality to link concepts intuitively. These aspects make Semantic UI so simple that even developers with very little coding experience will feel fairly comfortable and confident in using it. Another notable feature of Semantic UI is the integration with a dizzying array of third-party libraries, making the development process easier and more streamlined.

The developers that seek for a more complex design and development may find Semantic UI lacking. Also, the huge size of packages (806 KB) when compared to Bootstrap could lead to some displeasure.

3.7.3 Materialize

Materialize\(^{25}\) is a responsive front-end development framework that implements material design specifications and it is loaded with buttons, icons, cards, forms and other components. Created and designed by Google, material design is a design language that combines the classic principles of successful design along with innovation and technology. The main goal is to develop a system of design that allows for a unified user experience across all Google products on any other platform.

This framework includes a convenient grid feature that can be used for website layouts and it is also loaded with out of the box CSS components for material design shadows, typography, colors and other features. The wide range of components and features provided by Materialize, makes his file so large that could be a bulky framework to work with.

<table>
<thead>
<tr>
<th>Frameworks</th>
<th>Main Focus</th>
<th>Responsive Design</th>
<th>CSS Preprocessors</th>
<th>Prototypes</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>reTHINK</td>
<td>Protocol independent communication applications</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>6.8 MB (Runtime)</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>Web application design</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>154 KB</td>
</tr>
<tr>
<td>Semantic UI</td>
<td>Web application design</td>
<td>Yes</td>
<td>Limited</td>
<td>Yes</td>
<td>806 KB</td>
</tr>
<tr>
<td>Materialize</td>
<td>Web application design</td>
<td>Yes</td>
<td>Limited</td>
<td>Yes</td>
<td>931 KB</td>
</tr>
</tbody>
</table>

This comparative analysis leads to the conclusion that the views or the UI is completely out of scope of the reTHINK framework. This does not mean that it is not possible to use an additional library to handle

\(^{24}\)https://semantic-ui.com/, last accessed July 17th, 2017
\(^{25}\)http://materializecss.com/, last accessed July 17th, 2017
with the views at the same time that the reTHINK framework. The developer will have an additional
development effort to integrate it in the application.

Another important information that we can realize based on the Table 3.4 is that the reTHINK frame-
work is significant larger, which can be a disadvantageous, as we shall see in the evaluation chapters.

3.8 Chapter Summary

This Chapter analyses and discusses on how to choose the most well-suited framework for web ap-
plication development, taking into account the features that they usually provide to facilitate the rapid
development. In addition, we presented an overview on how to elaborate a methodology to evaluate
frameworks for web application development and what are the most relevant criteria.

Then, we provide a full description of the reTHINK framework since its architecture until a complete
example of an operation with the framework. This description appears early in the beginning because
we wanted to compare the reTHINK framework with all types of frameworks described afterwards.

Each comparative analysis allowed us to take several conclusions, among which we can highlight
the fact that the reTHINK framework should not be considered a full-stack framework, but it is much
more complete in terms of built-in features than a typical non-full-stack framework such as a WebRTC
framework. In addition, the views are out of scope of the reTHINK framework, so the developers should
handle it separately.
Chapter 4

Proposed Evaluation Methodology

This chapter describes the evaluation methodology that will be used to evaluate the reTHINK framework from the point of view of ease of developing applications when compared to traditional technologies, testing the benefits and the costs in terms of complexity.

Firstly, we describe in detail the methodology used to evaluation the reTHINK framework. This methodology consists in the development of a web application and an evaluation considering several points of view. Afterwards, we present the main target features that we want to use, test and evaluate considering the most important use cases targeted by the reTHINK framework. Then and taking into account those features, we define an application and its functional and non-functional requirements. We provide two different architectures for the same proposed application to allow the comparison between both developments, i.e. with reTHINK framework and with traditional technologies.

4.1 Methodology

The methodology used to evaluate the reTHINK framework from the point of view of ease of developing applications when compared to traditional technologies, consists in the development of a web application and an evaluation considering several points of view.

The application to develop was defined taking into account the most important use cases targeted by the reTHINK framework. To be able to compare the development using the reTHINK framework with the development using traditional technologies, we propose the implementation of two versions of the same application. One version will be implemented using only the features provided by the reTHINK framework, while the other version will be developed using traditional technologies (e.g. frameworks, libraries, databases). In each implementation, we will try to understand what are the advantages, disadvantages and limitations of each technology.

Completing these implementations, we will apply an evaluation composed by three different perspectives: the development evaluation, the applications evaluation and, the users’ and developers’ evaluation. Firstly, we have the evaluation from the development perspective. In this evaluation, we will use several criteria such as documentation, development time, programming effort and ease of installation and
deployment. All the data for this evaluation will be collected from the development of both applications. The goal of this evaluation is to compare the ease of development using both technologies.

The second part of this evaluation will be the applications evaluation. In this evaluation, we will verify if it was possible to achieve all the functional requirements in both developed applications. In addition, performance and portability tests shall be conducted to the develop applications. For example, testing the efficiency of the discovery of users (performance) and the behavior of the developed applications in different browsers (portability). The aim with this second evaluation is to compare the benefits and limitations of the used technologies.

Finally, the evaluation from the users’ and developers’ perspective will allow us to complete an overall analysis about the used technologies. Firstly, we are going to do some tests with users, regarding the usability of the developed applications to gather some initial feedback about the used technologies and features. And then, to make this evaluation more robust, we will conduct some questionnaires on some programmed events, where experienced programmers will have the opportunity to use the reTHINK framework to develop their own web applications.

Considering the obtained results from this evaluation, we will present an overall analysis and important recommendations to the reTHINK project. We think that following these recommendations will make the framework more usable and efficiency in the future.

### 4.2 Main Target Features

Since the ultimate goal of the reTHINK project is to provide an alternative model to the currently dominant walled communication networks, we must use communications capabilities, in particular chat capabilities. Another very important aspect is that reTHINK was design to be WebRTC compliant as much as possible. With this in mind, we need to use and test the WebRTC features, which will do through video and audio calls between clients.

In addition, to define an application capable of evaluating and characterizing the reTHINK framework, it is necessary to gather its main target features. After some analyses and taking into account the section 3.3, we consider that the most relevant features are:

- **Protocol independent communication**: applications do not have to specify which communication protocol to use. The reTHINK framework deals with the communication for each application depending on the type of service;

- **Identity portability and management**: the reTHINK framework empowers users with the choice and the management of their private data and identities. For example, users can choose the preferred IdP for authentication (among those available);

- **Discovery service**: users can search and communicate with other users without the need to be a subscriber of a certain communication service.
4.3 Application Proposal

In order to cover the most important use cases targeted by the reTHINK framework and its main features, we defined a video conference application. This application should allow users to create chat rooms to communicate with others through messages. In addition, users can also make video or audio calls within or outside the context of a chat room. Users should be authenticated using their preferred IdP and after authentication, each user should have access to their profile, created using the information (e.g. name and email) retrieved from the IdP used.

Each chat room should be identified with a certain name that may not be unique. When a chat room is created, it should be provided with a unique URL, which other users can use to join the chat room. In addition, users should be allowed to invite others to the chat room only if they are the owners. This invitation may happen in two different moments: when the user is creating the chat room or when the chat room is already created. Users can be invited by others through their email address to a chat room or video call, and the invited users should have the ability to accept or reject the invitation.

The available users and all type of events such as users entering, leaving or closing the chat room should be displayed inside a chat room. If the owner closes the chat room, all the users are automatically removed.

4.3.1 Requirements

Considering the proposed application and its features, it is now possible to structure its functional and non-functional requirements. Starting with the functional requirements, the proposed application should provide:

- Authentication system: users should be authenticated using their preferred IdP. In addition, it should be possible to retrieve profile information from the IdP used;

- Creation of chat rooms: users can create chat rooms to communicate with others through messages. The owner of the chat room can invite other users through their email addresses or they can use the unique chat room URL to join the chat room;

- Establishment of video calls: users can make video calls using the email address of the user that they want to communicate. The invited user should have the possibility to accept or reject the call;

Moreover, the application must fulfill the following non-functional requirements:

- Performance: the application must present a good performance, depending on the context. For example, the users should be able to exchange messages quickly and the invitation process should be as efficient as possible;

- Usability: the application should have a good usability. This means that the specified consumers can achieve quantified objectives with effectiveness, efficiency, and satisfaction in a quantified context of use;
- Portability: the application should run in different environments, e.g. different browsers. This requirement is particularly important because the WebRTC standard is supported differently by browsers, which means that some functional requirements will work differently or may even not work properly.

4.4 Architecture

To minimize subjectivity in this evaluation, we will develop two versions of the same proposed application using different technologies. This decision allows the comparison of both developments and, consequently, obtaining certain conclusions.

However, the difference between the used technologies (i.e. the reTHINK framework and the State of the Art) and considering the requirements presented on Section 4.3.1, requires the definition of two different architectures. One architecture for the development with traditional technologies (or as we call using the State of the Art of this kind of technologies), and another architecture for the development with the reTHINK framework. The order in which both architectures appear in this section is intentional because we wanted to highlight the main differences between them, especially, which components are useless and which are still needed.

4.4.1 Supported by the State of the Art

Taking into account the Figure 4.1, this architecture includes five main components: a web server, a database, an IdP, a client and an ICE server.

The web server is probably the most important component in this architecture because of its responsibilities. This component must handle the HTTP and WebSocket requests from clients, e.g. for rendering web pages, in case of HTTP and for searching users, in case of WebSocket. In addition, the web server should use cookies to know who connects to the server, for tracking an application usage and to deliver the dynamic content requested. To authenticate users through external IdPs, this component needs to integrate an authentication middleware that supports this type of authentication. A WebRTC API should also be used to deal with the WebRTC capabilities of the application (e.g. exchange of messages and video calls). Regarding the server side, it should integrate an API capable of handling the signaling process, essential in the WebRTC standard. Having all these APIs integrated in the web server should facilitate the rapid development and lower the programmer effort.

Another component in this architecture is the database. Database systems are a ubiquitous and critical component of many modern applications. Therefore, it is a component that must be selected taking into account several factors such as performance, availability and scalability. Although, the database system is not the focus of this work, because the applications will be used only for testing purposes, its choice is an important decision. The first big decision while selecting such system is whether use a relational (SQL) or NoSQL database.
Based on the requirements presented in Section 4.3.1, we needed a system for storing the application data, such as profile information for discovery purposes, chat room and accounts information. With this in mind, the system should provide high availability and a dynamic schema without the need to have the structure and data types fixed in advance. In addition, the data manipulation should be through object-oriented APIs, i.e. ORMs, to easily query and manipulate data without writing vendor-specific integration code. With this in mind, and taking into account the Table A.1, we chose to use a NoSQL database.

The third component is the IdP, which could be more than one, e.g. Google, Microsoft and Facebook. This component allows the authentication of users and retrieval of profile information about the user, such as name, email and profile image. Although this component is represented in this architecture, the authentication middleware integrated in the web server should handle it without further development.

Then, we have the client that runs in each user’s device. This component includes web pages obtained from the web server, a framework to deal with the UI and an WebRTC client API. This WebRTC client API should provide a common interface to deal with the WebRTC APIs such as RTCPeerConnection and RTCDATAChannel.

Finally, the last component is the ICE server, which could also be more than one. As already mentioned in Section 2.2.4, the ICE servers are mainly used as fallbacks in case a direct connection can not be established.

### 4.4.2 Supported by the reTHINK framework

The Figure 4.2 represent the second version of the architecture of the proposed application. This architecture was design to take full advantage of the features provided by the reTHINK framework. The aim of this design decision is to understand what are the limitations and benefits of the reTHINK framework and what we need, in addition, to comply with the requirements defined in Section 4.3.1.

This architecture includes six main components: a SP, a Global Registry, a Discovery Service, an
ICE server, an IdP and the clients. The SP provides a Message Node, a Domain Registry, a Catalogue and a web server. Among these components in the SP, only the web server is not provided by the reTHINK framework. This web server, unlike the previous architecture, is responsible only for providing the web pages to the clients. At this point, it is obvious that we need more components running on the SP, which raises the infrastructure costs and maintenance.

Then, we have the Discovery Service and the Global Registry. Both components are provided by the reTHINK framework and they need not be deployed by the SP. As already explained in Section 3.3.1, these two components allow a global discovery of users among several SPs.

The fourth component is the IdP. The reTHINK framework provides access to several IdPs such as Google and Microsoft. In addition, and following the available documentation, we hope that it will be easy for developers to add more IdPs.

Afterwards, we have the client that runs in each user device. In this architecture, this component includes web pages obtained from the web server, an API to deal with UI, two Hyperties deployed from the SP Catalogue and the Runtime Core. These Hyperties should provide the main features to the application. One Hyperty should deal with the chat capabilities such as creation of chat rooms and exchange of messages, while the other should deal with the WebRTC capability such as video calls between users.

Finally, the last component is the ICE server, which could be also more than one. This component has the same purpose that the previous architecture.

### 4.5 Chapter Summary

In this Chapter, we defined a web application capable to evaluate and characterize the reTHINK framework regarding its main target features. For this, we proposed two different architectures of the same
proposed application. In addition, to be able to evaluate the reTHINK framework from the point of view of ease in developing applications when compared with the traditional technologies, we proposed an evaluation divided into three parts.

The next two Chapters addresses the main decisions adopted regarding the implementation of the two versions of the proposed application. It starts with the video conference application supported by the reTHINK framework because we wanted to highlight the advantages and disadvantages of the reTHINK framework first, and then compare it with the video conference application supported by the State of the Art.
Chapter 5

Video Conference Application supported by the reTHINK framework

This chapter addresses the main decisions adopted regarding the implementation and configuration of the video conference application supported by the reTHINK framework. Firstly, we present all the technologies that were used in the development of each component presented in Figure 4.2. Afterwards the development process is described focusing on the main requirements of the application.

5.1 Software Choices

Taking into account Section 4.4.2, the SP components were all provided by the reTHINK framework except the web server. As we had to develop new Hyperties, we replaced the Catalogue for a tool called Hyperty Toolkit as suggested by the reTHINK documentation. This tool acts as a Catalogue, encoding in real-time all the changes made in the Hyperties development process. The Domain Registry¹, Message Node² and Hyperty Toolkit³ were all deployed from each official repository, following the available documentation. The web server used is a simple Node.js server that only exposes the HTML and JavaScript files of the web application.

On the other hand, as the Discovery Service and Global Registry had to be deployed in different machines, we choose to use the Discovery Service⁴ provided by the Deutsch Telekom (partner of the project) and we deployed one instance of the Global Registry⁵ in an INESC-ID’s server (partner of the project). We choose to use the available instance of the Discovery Service because its documentation is very complex and difficult to follow. Additionally, the Discovery Service is supposed to be the entry point for new developers to the reTHINK framework, so there is no value in having more than one instance running.

¹https://github.com/reTHINK-project/dev-registry-domain , last accessed August 12th, 2017
²https://github.com/reTHINK-project/dev-msg-node-vertx , last accessed August 12th, 2017
³https://github.com/reTHINK-project/dev-hyperty-toolkit , last accessed August 12th, 2017
⁴https://rethink.tlabscloud.com/discovery/ , last accessed August 12th, 2017
⁵https://github.com/reTHINK-project/dev-registry-global , last accessed August 12th, 2017
For the client side and considering the study carried out in Section 3.7, we choose to use the Bootstrap framework to deal with the UI of the application. This framework is the most complete and has dozens of custom HTML and CSS components that allow for rapid development. In addition, the client side takes advantage of the JavaScript and JQuery technologies to make the UI more responsive and fully functional.

Finally, for the ICE servers, we chose to use public servers available online, as there are several STUN and TURN servers available.

5.2 Implementation

After choosing the technologies, we implemented each functional requirement defined in Section 4.3, starting with the authentication of users, then with the creation of chat rooms and finally with the establishment of video calls between users.

5.2.1 Authentication of Users

As already mentioned, for this application we developed two different Hyperties. One to deal with the chat rooms and exchange of messages, called Group Chat Manager, and another to deal with the video calls between users, called Connector. To successfully deploy these Hyperties from the Hyperty Toolkit in the developed application, authentication is mandatory because the user identity is essential in the registration of the Hyperty instance in the Domain Registry.

After an intense reading of the reTHINK documentation, we realize that each Hyperty needs to interact with the Service Framework library, namely Syncher, Discovery and Identity Manager components. These components will later interact with the Runtime Core components (this separation is due to sandboxing mechanism). In addition, to ensure the interoperability the only thing that needs to be standardized are the data schemas. As the reTHINK documentation suggests, we reuse the available data schemas for the developed Hyperties.

![Authentication form provided by the reTHINK framework](image)

Figure 5.1: Authentication form provided by the reTHINK framework

When the application starts to deploy these Hyperties, the authentication form provided by the re-
THINK framework is automatically shown, prompting the user to authenticate. As Figure 5.1 shows, the available IdPs are Google, Microsoft and Orange (partner of the project). For the developed application, we only use and tested the first two IdPs. Each option available, e.g. google.com, corresponds to an IdP ProtoStub that is responsible for handling the communication between the IDM and the IdP.

After authentication, the application should be able to obtain the identity object to fulfill the user profile (see Figure B.1). In this case, the application can obtain the identity object through the deployed Hyperty. Considering the Figure 5.2, the retrieved identity object is composed by an avatar (profile picture URL), a cn (common name), a locale, a userURL and a username (email) field. Although, the identity object provided was sufficient to complete the user profile, there are fields that raise some questions. For example, the common name is the decomposition of the email address, when it should be obtained directly from the IdP through the OAuth protocol.

```
{ "identity" : {
    "avatar" : "https://lh4.googleusercontent.com/-IXVn1ny9JBU/.../photo.jpg",
    "cn" : "rethink.inesc.test",
    "locale" : "pt-PT",
    "userURL" : "user://google.com/rethink.inesc.test@gmail.com",
    "username" : "rethink.inesc.test@gmail.com"
}
```

Figure 5.2: User identity object supported by the reTHINK framework

This application requirement was implemented using only and exclusively the features provided by the reTHINK framework, which allowed us to speed up the development with a low programming effort. However, this authentication system has some limitations that may not appeal to developers. Firstly, the authentication form is hardcoded, which means that the developer must use it, getting dependent on its UI, colors and usability degree. Secondly, this authentication system does not support the creation of accounts and anonymous identities because of its compliance with the WebRTC identity model. Thirdly, after associating an identity with the deployed Hyperty, if we try to replace the identity, it seems to not work properly because the Hyperty continues to use the previous identity. And finally, there are no documentation available about how to develop and add more IdP ProtoStubs, e.g. for Facebook.

### 5.2.2 Chat Rooms

The Group Chat Manager Hyperty is responsible for handling the creation of chat rooms, invitation of users, exchange of messages between users and the join of users in the chat rooms.

For this application, a chat room can be seen as a Data Object that is associated with the deployed Hyperty instance. The Hyperties communicate and cooperate with each other through a data synchronization model called Reporter - Observer. This model uses a P2P data stream synchronization solution for programmatic objects e.g. JSON objects. To avoid inconsistencies among peers, only one has writing permissions to the Data Object (Hyperty Reporter), while the others only have permissions to read (Hyperty Observer). As soon as the Reporter performs changes to the Data Object, they are immediately propagated to any authorized Observer (see Figure 5.3). A Runtime Core component called Syncher
is responsible for handling the requests to manage the Data Objects such as creation, subscription, unsubscription and receiving its updates, e.g. a new participant or a new message. All these events can be received by the Hyperties and, consequently, by the application.

However, the restriction that only one peer can write in the Data Object can be seen as a disadvantage, depending on the requirements of the application. For example, the Hyperty Observer does not have the ability to add new participants to the chat room (Data Object) without the permission of the Hyperty Reporter.

![Figure 5.3: Reporter - Observer model](image)

To successfully create a chat room, the user can associate a name and the email addresses of the users that he wants to invite to the chat room. Through the Discovery component (Service Framework library), the Group Chat Manager Hyperty can discover the registered Hyperties associated with those emails and invite them to join the chat room (see Figure B.2). If they accept the invitation, their Group Chat Manager Hyperty instance will subscribe to the Data Object (chat room) and act as an Observer. In addition, the users can join the chat room if they know it is unique URL. The Figure 5.4 shows a chat room example.

![Figure 5.4: Chat room example](image)

The reTHINK framework provides a global discovery of Hyperty instances across multiple SP domains using the Global Registry, Domain Registry and Discovery Service components. This global discovery process is a huge and unique advantage for those who use the reTHINK framework. Additionally, it is an important process to allow for interoperability.

However, after testing this global discovery feature we found some limitations, such as:
It seemed that the Global Registry was not very well integrated with the Runtime Core components because, and following the documentation, when we deployed those Hyperties we did not get the actual GUID. This GUID is essential in the creation of the user profile in the Discovery Service and later to resolve the queries. To overcome this limitation and test this feature, we had to manually populate the Global Registry;

In addition, each user must create a user profile in the Discovery Service (i.e. through the Web). Each user profile has some fields that users need to know such as their GUID and domains. Once more, we had to manually populate the Discovery Service with the user profiles for testing purposes;

The queries provided by the Discovery Service are somewhat limited, resulting in inaccurate results. The addition of some advanced queries would be a great improvement on this component, allowing to achieve better results. Suggested queries could be, e.g., by email, GUID and user interests.

These limitations show a lack of automation on this global discovery process, which definitely may not please the developers. With this in mind, for this application we used the discovery process restricted to the used SP domain, taking advantage only of the Service Framework library and Domain Registry.

To allow use cases like the exchange of messages between users of the same chat room, where the Observers need to have writing permissions, the reTHINK framework introduces the Child Data Objects. When a user sends a message, a Child Data Object is created in the Data Object associated with the chat room. This creation is consequently propagated to all the participants (Reporter and Observers) of the chat room. In this case, the created Child Data Object carries the message and the identity object of the sender (see Figure 5.5).

The chat rooms features (e.g. exchange of messages and invitation of users) were also implemented using only the features provided by the reTHINK framework. However, unlike the authentication of users, these features required a tremendous programming effort, especially at the Hyperty level. Firstly, the documentation is not well structured in order to facilitate the understanding of these complex concepts, e.g. Data Objects and Child Data Objects. And then, it is not very clear what libraries are needed and what methods should we use to achieve quantified goals.

We think that without the examples provided by the reTHINK framework, it would be very difficult to developed the Group Chat Manager Hyperty to deal with such requirements. But there are also advantages that should be highlighted. First, the Reporter - Observer pattern brings a lot advantages to the developer such as no need to implement an additional mechanism for messages delivery, a wide range of events available (e.g. new participant, new message, participant leaved and chat room closed) and no need to integrate an additional database system to save the application information. It is important to mention that because of the application requirements was not necessary to integrate an additional database system. The reTHINK framework saves all the application information needed for the proper function of the application, but it does not replace the need for a database system if it were necessary to save another data types (e.g. new accounts).
Secondly, the reTHINK framework provides a feature called Resume, that automatically loads the active Data Objects associated with the Hyperty and the exchanged messages. And finally, it is also possible to know the user availability (online or offline) through the Hyperty status with a very low programming effort.

5.2.3 Video Calls

The Connector Hyperty has the responsibility to deal with the establishment of video calls between users. In addition, it needs to handle the signaling process and every other issue that WebRTC involves.

This Hyperty, as suggested by the reTHINK documentation, uses the WebRTC Adapter.js\(^8\) that provides the all-in-one WebRTC solution cross-browser. The Adapter.js is an open source project that insulates applications from the changes of specifications. While the WebRTC specification is relatively stable, not all browsers have fully implemented all of its features. For each version of each browser that supports WebRTC, Adapter.js implements the needed polyfills, establishes the non-prefixed names of APIs, and applies any other change needed to make the browser run code written to the WebRTC specification. Besides that, it allows the management of the signaling events such as the reception of local and remote offers, the addition of new streams and the access to the status of the ICE candidates.

To start a video call between two users, the discovery process needs to take place in order to find the Hyperty instance associated with the email address of the invited user. After that, a Data Object is created following the same principles mentioned in the previous section. The only difference is the used data schema, because we are dealing with a different type of service.

Like the chat room feature, the invited user has the ability to accept or reject the call. If the invited user accepts the call, they will share a media stream of P2P data (video and audio) with each other and the call takes place (see Figure 5.6). On the other hand, if the user rejects the call, this rejection is not propagated to the caller and, consequently, he does not know about it. To overcome this problem, the invited user automatically accepts the invitation and then, if he rejects the call, it will be automatically

\(^8\)https://github.com/webrtc/adapter, last accessed August 15th, 2017
switched off. This problem seemed to be related with some bugs in the Service Framework library. We successfully reported these bugs to the reTHINK development team and they should be resolved in the near future.

The establishment of video calls was implemented using only the reTHINK framework and the Adapter.js. Although, its implementation was quite straightforward, we think that it required more effort than it would require with different technologies, as we will see in Chapter 6. At this point, we do not see any advantage of using the reTHINK framework and the Adapter.js for the implementation of this feature.

![Figure 5.6: Call supported by the reTHINK framework](image)

**5.3 Chapter Summary**

In this Chapter, we described the technologies used in the development of the video conference application supported by the reTHINK framework. In addition, we explained in detail the implementation of the functional requirements of the application, highlighting the limitations found, and the advantages and disadvantages of using the reTHINK framework. Regarding the non-functional requirements, the used technologies should provide good performance, portability and usability. In the Chapter 7, we will perform some performance and portability tests. The usability will be evaluated using tests with users. It is important to mention that because of the similarity of both developed applications in terms of UI, the usability will be evaluated on a higher level. The aim is to receive some initial feedback about the used technologies and certain features in terms of UI (e.g. authentication form).

At the end, we were able to develop the proposed application, covering all the defined requirements.
Chapter 6

Video Conference Application supported by the State of the Art

This chapter addresses the main decisions adopted regarding the implementation and configuration of the video conference application supported by the State of the Art, which means that it was developed with traditional technologies that a typical programmer would use to develop this type of web application. Firstly, we present all the technologies used in the development of each component presented in Figure 4.1. Afterwards the development process is described focusing on the main requirements of the application.

6.1 Software Choices

Considering Section 4.4.1, we decided to choose first the WebRTC API that will be integrated in both the server side and the client side. This API should deal with the signaling process and allow the transfer of P2P data streams (messages and video). Taking into account the study carried out in Section 3.6, we choose to use the PeerJS framework because it seemed to be very simple to use, well documented and compatible with the most popular web browsers. In addition, the same public ICE servers, as for the previous application, were used.

After choosing the WebRTC API, we needed a web server to deal with the HTTP and WebSocket requests, and identification of users. This web server would have to be compatible with the PeerJS server, allowing the interaction between them. The PeerJS server documentation suggests the use of the Express web server. Our study about non-full-stack frameworks (see Section 3.5) revealed that the Express framework should be enough to implement all the application requirements, so we choose to use the Express web server.

Then, we needed to select a database system to storage the application data such as user accounts and chat rooms information. Adding the capability to connect databases to Express-based web applications is just a matter of loading an appropriate Node.js driver. The Express documentation sug-
gests several modules for database systems. We choose to use MongoDB\(^1\) because it is a NoSQL
document-oriented database that uses JSON-like documents with schemas, providing high availability
and scalability. In addition, we successfully integrated a module called Moogose\(^2\) that is an ORM for
MongoDB, allowing to manipulate data easily.

The last component chosen for the server side was the authentication API. We needed an API
that allows users to authenticate using their preferred IdP and, at the same time, compatible with the
Express web server. Using, our study about full-stack frameworks (see Section 3.4), we noticed that
some frameworks use the Passport.js, which is a authentication middleware for Node.js. After a deep
reading of its documentation, we conclude that Passport.js is perfect for our application not only because
it can be unobtrusively dropped into any Express-based web application, but also because it provides
several authentication protocols (e.g. OAuth and OpenID Connect) and works with several IdPs such as
Google and Facebook.

To deal with the UI of the developed application, the Bootstrap framework, JavaScript and jQuery
technologies were also used. We decided to use the same technologies for the client side because it
is not in the scope of the reTHINK framework (the view), so these technologies will not be particularly
tested or evaluated.

Additionally, we have integrated other modules in the Express web server to deal with minor features
as shown in Table 6.1.

<table>
<thead>
<tr>
<th>Modules</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Express Session</td>
<td>Storage the session data.</td>
</tr>
<tr>
<td>Pug</td>
<td>Template engine for rendering web pages.</td>
</tr>
<tr>
<td>Connect-Flash</td>
<td>Used for storing session messages, e.g. error messages.</td>
</tr>
<tr>
<td>Cookie-Parser</td>
<td>Middleware that enables signed cookies.</td>
</tr>
<tr>
<td>Socket.IO</td>
<td>Allow the communication between the web server and the clients via WebSockets.</td>
</tr>
<tr>
<td>Bcrypt</td>
<td>Password hashing library.</td>
</tr>
</tbody>
</table>

### 6.2 Implementation

After choosing the technologies, we implemented each functional requirement defined in Section 4.3.1,
starting with the authentication of users, then with the creation of chat rooms and finally with the establish-
ishment of video calls between users.

\(^1\)https://www.mongodb.com/, last accessed August 16th, 2017
\(^2\)http://mongoosejs.com/, last accessed August 16th, 2017
6.2.1 Authentication of Users

With Passport.js, the authentication of users using their preferred IdP is just a matter of loading the appropriate authentication strategy. The first decision that we had to do was select the IdPs to use in the application. The reTHINK framework works with Google, Microsoft and Orange IdPs, but we used Facebook and Google because they are more appealing to users. To configure these IdPs, we only need to define the routes, callbacks and developer keys in the web server. In addition, and contrary to the application support by the reTHINK framework, we had to develop from scratch the authentication form that is prompted and shown to the users (see Figure 6.1).

When we started to develop the authentication form, we had to deal with our first problem. In the beginning, we started using the Handlebars\(^3\) to deal with the rendering of web pages. However, the version of this template engine for Express was deprecated, resulting in several internal errors. To solve this, we changed our software decision and we started to use another template engine called Pug\(^4\). We chose Pug because it is well supported by Express and it provides very good documentation.

![Developed authentication form](image)

**Figure 6.1: Developed authentication form**

After authentication, we can retrieve from the used IdP the name, email, profile ID, profile picture URL and token, through the OAuth protocol (see Figure B.3). This retrieved information forms a unique account data model that it will be saved in MongoDB through the Moogose API (see Figure 6.2).

```json
{ "account": {
  "email": String,
  "id": String,
  "chat": [{"name": String, "url": String, "ownerEmail": String}],
  "facebook": {"token": String, "email": String, "name": String, "photo": String, "active": Boolean},
  "google": {"token": String, "email": String, "name": String, "photo": String, "active": Boolean},
  "local": {"name": String, "email": String, "password": String, "active": Boolean}
}
```

**Figure 6.2: Account data model**

In addition, with this application we wanted to go further and show how easy it is to add new authentication schemas through the creation of local accounts. To implement this additional requirement, we used a strategy provided by Passport.js and the Bcrypt\(^5\) library to ensure higher security in the storage.

---

\(^3\)http://handlebarsjs.com/, last accessed August 17th, 2017

\(^4\)https://pugjs.org, last accessed August 17th, 2017

\(^5\)https://www.npmjs.com/package/bcrypt, last accessed August 17th, 2017
of passwords and their validation.

Although the implementation of this authentication system was straightforward, it required a lot more programming effort than that required for the application supported by the reTHINK framework. In this application, we had to develop the web server from scratch, integrate a group of APIs and an authentication form. However, there are a few advantages that should be highlighted. First, the programmer is free to choose the authentication schemas for the application and it is easier to add new authentication schemas or new IdPs in the future. Secondly, the programmer has the responsibility to develop an authentication form, which allows greater customization and usability. Lastly, the ability to create local accounts represents a very important advantage because there may be users who do not wish to use any of the available IdPs or do not have accounts.

6.2.2 Chat Rooms

The main goal of the chat rooms is to enable the exchange of messages between users. This ability is possible through the use of the PeerJS framework. As already mentioned in Section 3.6.1, each peer must have a unique identifier. In our implementation, the web server is responsible for generating this identifier because we wanted to ensure its uniqueness and save it in database for discovery purposes.

In this application, a chat room is a simple JSON object containing its name, URL and owner email. Every time, a user wants to initiate a chat room, a request to the server is sent through the Socket.IO6 library for generating a unique URL to allow the joining of other users.

To be able to invite users, exchange messages and start video calls, each user must create his own peer object (see Figure 6.3). Instead of discovering Hyperty instances to invite other users to the chat room, now it is necessary to discover the peer identifiers associated to the email addresses. The web server is the component responsible to perform this discovery process. After that and using the discovered peer identifiers, the user can start a connection with them (see Figure B.4). With the connection established, the user can use the data connection to exchange messages or to receive certain types of events such as a peer leaving the chat room and the joining of other user to the chat room.

```javascript
peer = new Peer(peerID, {
    "host": HOSTNAME,
    "port": PORT,
    "path": "/peerjs-server",
    "debug": 3,
    "config": {
        "iceServers": [
            {
                "url": "stun:stun1.l.google.com:19302"
            },
            {
                "url": "turn:turnnumb.viagenie.ca",
                "username": "webrtc@live.com",
                "credential": "muazkh"
            }
        ]
    }
});
```

Figure 6.3: Peer configuration

The chat one-to-one user was quite simple to implement, but when we started the implementation of multi chat feature (many users), we had some problems. After reading the PeerJS documentation, we did not find any solution or suggestion for the implementation of this feature. With the aim to find some

6https://socket.io/, last accessed August 17th, 2017
useful answers, we discussed this question on their official GitHub repository and we encountered one possible solution. The peer, owner of the chat room, will be responsible for broadcasting, to all his active connections, all the events and messages received (see Figure 6.4). Although, we recognized that this solution is not ideal because it creates some inconsistencies among peers, it allowed us to implement this requirement. In fact, this solution is very similar, in terms of concept, to what is used in the reTHINK framework.

The well-structured documentation of PeerJS framework allowed us to implement this requirement with a much smaller programming effort. Although, we had to develop several components from scratch, design solutions to their interactions and integrate additional libraries to deal with several aspects, we were very pleased with the final result.

However, there are disadvantages and limitations that should be highlighted. First, when compared to the reTHINK framework, the PeerJS framework is very limited in terms of events related to the active connections. To have the same type of events (but not all), we had to add additional metadata to the connections. Secondly, there is no Resume feature in the PeerJS framework, which means that when a user refreshes the web page or leaves the application, all the connections and their information will be lost. To solve this, the application would need to be constantly loading the chat rooms information from the database. And finally, there are some security concerns that should be considered. Although, this application was developed only for testing purposes, the communication between the web server, PeerJS server and clients would need to be better secured, e.g. using HTTPS and certificates.

6.2.3 Video Calls

With the peer object configured, the client can start a video call whenever he wants, it is just a matter of dealing with the corresponding events. In this case, a call event is issued every time a user starts a video call. The invited user has the ability to accept or reject the call offer. If he accepts the invitation, the call proceeds and they will share a P2P data stream with each other (see Figure 6.5). In addition, there is no need to waste time with the signaling process or other issues because the active PeerJS server handles with all these questions.
The implementation of this last application requirement was very simple, with a lot less programming effort than that required for the application supported by the reTHINK framework. At this stage, we do not see any advantage in using the reTHINK framework to deal with the establishment of video calls between users. The API provided by the PeerJS framework is simple and intuitive and we did not have any problem in the implementation of this application requirement.

6.3 Chapter Summary

In this chapter, we described the technologies used in the development of the video conference application supported by the State of the Art. In addition, we explained in detail the implementation of the functional application requirements, highlighting the limitations found, and the advantages and disadvantages of using such technologies. Regarding the non-functional requirements, the used technologies should provide good performance, portability and usability. In the Chapter 7, we will perform some performance and portability tests. The usability will be evaluated using tests with users. It is important to mention that because of the similarity of both developed applications in terms of UI, the usability will be evaluated on a higher level. The aim is to receive some initial feedback about the used technologies and certain features in terms of UI (e.g. authentication form).

Despite that we tried to make a comparison between this development with the development using the reTHINK framework. In fact, we saw that this development required less programming effort and the concepts were easier to understand. Although, there are advanced features provided by the reTHINK framework that were not possible to achieve with the technologies used in this development. This comparison between developments will be better exploited in the next chapter.
Chapter 7
Evaluation and Comparison Analysis

This chapter addresses the evaluation carried out in this work. The aim is to evaluate the reTHINK framework from the point of view of ease in developing applications when compared with the traditional technologies, testing the benefits and the costs in terms of complexity. In addition, we wanted to validate some statements and conclusions made in the previous implementation sections.

To collect data from different perspectives, we divided this evaluation into three parts (development, applications and, users and developers evaluation) that are described in the following sections. At the end, we provide an overall analysis and a few recommendations considering all the conclusions reached about the reTHINK framework in each part of this evaluation.

7.1 Development Evaluation

The first part is the development evaluation. In this part, we tried to gather the most relevant metrics to evaluate the used technologies from the point of view of ease in developing applications.

Taking into account Section 3.2, we choose the documentation quality, programming effort, development time and, ease of installation and deployment as the most appropriate metrics for this part of the evaluation. All the data used in each metric was gathered from both developments of the video conference application.

7.1.1 Documentation

The documentation is often key to the success of the framework or any other type of technology. A good documentation should be simple to understand, providing plenty of examples, snippets, sample code, articles and tutorials. Well explained, detailed documentation always attracts new programmers or convinces those who are still in doubt between several technologies.

For this metric, we measured the time that we spent reading the documentation of the used technologies in each application development. In addition, we counted how many times throughout each development, it was necessary to resort again to the documentation to solve any issue or to better understand some concept.
Taking into account the data presented on Table 7.1, we can draft some conclusions about the documentation of the used technologies, specially about the reTHINK framework documentation.

Table 7.1: Documentation analysis

<table>
<thead>
<tr>
<th>Applications</th>
<th>Requirements</th>
<th>APIs</th>
<th>Time spent reading the documentation</th>
<th>How many times it was necessary to refer back to the documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video conference application supported by the reTHINK framework</td>
<td>Authentication System</td>
<td>reTHINK</td>
<td>3 h</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Chat Rooms</td>
<td>reTHINK</td>
<td>6 h</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Video Calls</td>
<td>reTHINK Adapter.js</td>
<td>3 h</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Authentication System</td>
<td>Express Pug</td>
<td>4 h</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cookie-Parser</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connect-Flash</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Moogose</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bcrypt</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Passport.js</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Chat Rooms</td>
<td>Socket.IO</td>
<td>2 h</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PeerJS</td>
<td>1 h</td>
<td>2</td>
</tr>
<tr>
<td>Video conference application supported by the State of the Art</td>
<td>Authentication System</td>
<td>Express Pug</td>
<td>4 h</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cookie-Parser</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connect-Flash</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Moogose</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bcrypt</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Passport.js</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Chat Rooms</td>
<td>Socket.IO</td>
<td>2 h</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PeerJS</td>
<td>1 h</td>
<td>2</td>
</tr>
</tbody>
</table>

Looking at the number of APIs used in both developed applications, it would be expected that the time spent reading the documentation for the application supported by the reTHINK framework would be shorter. However, it was necessary to spend 12 hours, which is approximately 42% more time than that spent with the other developed application. In our opinion, this is due to the unstructured documentation that the reTHINK framework provides, when compared with the documentation of other APIs. Currently, the programmers that want to use the reTHINK framework will find its documentation only on GitHub. This would not be a problem if the documentation was not spread by its 36 repositories, which makes its reading and understanding extremely hard.

In addition, it was necessary to resort again to the reTHINK documentation 21 times. Although, this represent an increase of 52% when compared with the application supported by the State of the Art, it can be explained by the new and complex concepts introduced by the reTHINK framework such as Hyperties and Data Objects.

It is important to mention that Table 7.1 does not include the time spent with the UI of the applications, because we used the same technologies and UI in both developed applications. Additionally, as already mentioned, we had participated in the development of the reTHINK project, so when we developed the application supported by the reTHINK framework, we had already a few months of knowledge about the framework and its concepts. Nevertheless, it was the first time that we develop a web application with the reTHINK framework.

7.1.2 Programming Effort

The goal of using a framework for web application development is to reduce the programming effort by not having to write everything from scratch. This is an important metric for developers when they are selecting the development technologies.

In the context of this work, both developed applications use technologies that provide features already
tested and working. For example, the reTHINK framework provides an authentication form and the Data Object mechanism, whereas the PeerJS framework allows an easy use of the WebRTC technology to exchange messages and start video calls between users. The aim of Table 7.2 is to quantify the programming effort in both developments, showing how many files we had to use and how many lines of code we had to program.

Table 7.2: Programming effort overview

<table>
<thead>
<tr>
<th>Applications</th>
<th>Number of files</th>
<th>Code lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video conference application supported by the reTHINK framework</td>
<td>12</td>
<td>2807</td>
</tr>
<tr>
<td>Video conference application supported by the State of the Art</td>
<td>12</td>
<td>2022</td>
</tr>
</tbody>
</table>

In both developed applications, we had to use and code the same number of files. However, the application supported by the reTHINK framework required more 20% code lines than the application supported by the State of the Art. We think that this difference (785 code lines) is not relevant enough to be taken into account in the developer’s decision. Most of it is due to the Hyperties development, where we had to program all the interactions with the reTHINK components using different libraries.

With this in mind, we believe that if the reTHINK project adopts a different business model that this will increase its potential and lower the programming effort. In this business model, the SPs (e.g. Orange) would be responsible for the Hyperty development and deployment, depending on the developer’s needs. After that, the developers just have to arrange with the SP to have full access to those Hyperties and their documentation. With this new approach, the application supported by the reTHINK framework would only require 4 files and 1399 code lines, which represents a huge reduction. Of course, with this new approach would be necessary to review other aspects such as performance and availability (e.g. Hyperties download and messages delivery).

7.1.3 Development Time

The third metric used in this part of the evaluation was the development time. The main goal of using a framework or another API is to lower the programmer effort by providing all the necessary features to develop the application. If this does not happen, then the API used is not well suited. Table 7.3 shows in detail how much time we spent in each application requirement for both developed applications.

Taking into account Table 7.3, we can see that the development of the application supported by the reTHINK framework took longer than the development of the application supported by the State of the Art. This situation occurs in practically all the application requirements, except with the requirements related with the authentication of users, because almost everything is provided by the reTHINK framework (e.g. authentication system, form and IdPs configuration).

However, the time spent in the development of the chat rooms and video calls in the application supported by the reTHINK framework was not the expected. We think that it was a consequence of the unstructured documentation provided by the reTHINK framework. As developers, we had to wonder across 36 repositories related with reTHINK every time we wanted to implement some feature. Although,
Table 7.3: Development time breakdown

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Video conference application supported by the reTHINK framework</th>
<th>Video conference application supported by the State of the Art</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication through IdPs</td>
<td>3 h</td>
<td>4 h</td>
</tr>
<tr>
<td>Local accounts</td>
<td>NA</td>
<td>2 h</td>
</tr>
<tr>
<td>Retrieve profile information</td>
<td>2 h</td>
<td>2 h</td>
</tr>
<tr>
<td>Chat rooms creation</td>
<td>8 h</td>
<td>4 h</td>
</tr>
<tr>
<td>Invitation of users</td>
<td>4 h</td>
<td>3 h</td>
</tr>
<tr>
<td>Join of users</td>
<td>3 h</td>
<td>2 h</td>
</tr>
<tr>
<td>Exchange of messages</td>
<td>1.5 h</td>
<td>1 h</td>
</tr>
<tr>
<td>Video calls</td>
<td>5 h</td>
<td>3 h</td>
</tr>
<tr>
<td>Call management</td>
<td>1.5 h</td>
<td>2 h</td>
</tr>
<tr>
<td>Total</td>
<td>28 h</td>
<td>23 h</td>
</tr>
</tbody>
</table>

the reTHINK framework provides concrete examples to implement some functionalities (e.g. exchange messages), it would be very good if comprehensive examples were provided, such as how to create a Data Object and what APIs the developer should use to manage it.

Nevertheless, if the approach presented on Section 7.1.2 was adopted, the development time of the chat room and video call features would not take into account the time spent with the Hyperties development (see Table 7.4). As we can see, the development of these requirements is now shorter and smaller than the development time of the application supported by the State of the Art.

It is important to mention that Tables 7.3 and 7.4 do not take into account the time spent with the development of the UI of both developed applications (approximately 12 hours) because both are similar and the same technologies were used.

Table 7.4: Development time breakdown without Hyperty development

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Video conference application supported by the reTHINK framework</th>
<th>Video conference application supported by the State of the Art</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chat rooms creation</td>
<td>4 h</td>
<td>4 h</td>
</tr>
<tr>
<td>Invitation of users</td>
<td>2 h</td>
<td>3 h</td>
</tr>
<tr>
<td>Join of users</td>
<td>1 h</td>
<td>2 h</td>
</tr>
<tr>
<td>Exchange of messages</td>
<td>0.5 h</td>
<td>1 h</td>
</tr>
<tr>
<td>Video calls</td>
<td>2.5 h</td>
<td>3 h</td>
</tr>
<tr>
<td>Call management</td>
<td>1 h</td>
<td>2 h</td>
</tr>
<tr>
<td>Total</td>
<td>11 h</td>
<td>15 h</td>
</tr>
</tbody>
</table>

7.1.4 Ease of Installation and Deployment

The last metric used in this part of the evaluation was the ease of installation and deployment of both the developed applications. This metric plays a very important role while selecting a framework or another API. Although, an API can be quick to learn, it can represent a problem if the developer has to get through several steps just to get it installed and working. This would also pose a big problem once the application is ready, tested and needs to be deployed to the production server.
For this metric, we collected for each developed application, the number of dependencies to install, steps until installation (commands), size (megabytes) and average installation time (seconds). In addition, we analyzed the ease of deployment for each developed application, namely the additional configurations.

The Table 7.5 shows that the video conference application supported by the reTHINK framework takes more time and requires more steps to install, is bigger and has much more dependencies than the other developed application. Although, the average installation time and the number of dependencies are, for us less relevant, we think that the steps until installation and the size should be considered.

The steps until installation are a consequence of the number of components, all initialized separately with different tools, e.g. Domain Registry with Docker, web server and Hyperty Toolkit with Node and Message Node with Maven. Additionally, the developer needs to change some configurations in different files and components. These steps could be improved a lot with the use of the same tool and initialization file for all the reTHINK components.

The size is also worrying but natural, given the 95 dependencies. After a quick look into it, we conclude that more than a quarter of the dependencies are for testing purposes. We think that with a revision, this number could be lower and consequently, so would the size of the components essential to run the application supported by the reTHINK framework.

The deployment is also a very important aspect when dealing with web applications. The application supported by the State of the Art does not require any additional configuration. However, the deployment of the application supported by the reTHINK framework could require more effort. Even though the ease of installation of the application supported by the reTHINK framework is improved by following the suggestions mentioned above, this application requires additional configurations such as the URL where the Message Node is running or the Domain Registry. In addition, and as a consequence of the number of components and its functionalities, this application requires more from the production server, increasing its costs. One possible improvement would be to automate the additional configurations needed for deployment.

Lastly, it is important to mention that if the approach presented on Section 7.1.2 was adopted, the ease of installation and deployment of the application supported by the reTHINK framework would improve a lot because it would used the components provided by the SP (i.e. Domain Registry, Hyperty Toolkit or Catalogue and Message Node). For example, the number of dependencies would be reduced to 4 and the size for approximately 39 MB.

Table 7.5: Ease of installation metrics

<table>
<thead>
<tr>
<th>Applications</th>
<th>Components</th>
<th>Dependencies</th>
<th>Steps</th>
<th>Size</th>
<th>Average installation time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video conference application supported by the reTHINK framework</td>
<td>Hyperty Toolkit, Domain Registry, Message Node, Web Server</td>
<td>95</td>
<td>11</td>
<td>680 MB</td>
<td>174 s</td>
</tr>
<tr>
<td>Video conference application supported by the State of the Art</td>
<td>Web Server, MongoDB</td>
<td>15</td>
<td>4</td>
<td>70 MB</td>
<td>52 s</td>
</tr>
</tbody>
</table>

The deployment is also a very important aspect when dealing with web applications. The application supported by the State of the Art does not require any additional configuration. However, the deployment of the application supported by the reTHINK framework could require more effort. Even though the ease of installation of the application supported by the reTHINK framework is improved by following the suggestions mentioned above, this application requires additional configurations such as the URL where the Message Node is running or the Domain Registry. In addition, and as a consequence of the number of components and its functionalities, this application requires more from the production server, increasing its costs. One possible improvement would be to automate the additional configurations needed for deployment.

Lastly, it is important to mention that if the approach presented on Section 7.1.2 was adopted, the ease of installation and deployment of the application supported by the reTHINK framework would improve a lot because it would used the components provided by the SP (i.e. Domain Registry, Hyperty Toolkit or Catalogue and Message Node). For example, the number of dependencies would be reduced to 4 and the size for approximately 39 MB.
7.2 Applications Evaluation

The next part of this evaluation is the applications evaluation. In this part, we used both developed applications to perform some performance and portability tests. Performance tests are important to developers because they want to have the opportunity to develop the most efficient application and different technologies can provide different performance levels. In addition, the WebRTC technology is so recent that not all browsers support it and even those that do can have different behaviors, harmful to the proper functioning of the application. In order to validate this, portability tests were carried out on both developed applications using different browsers. The portability tests are especially important because they allow to analyze the behavior of the reTHINK framework in different browsers (e.g. its installation, operation and Hyperties deployment).

To perform these tests, we setup the most realistic test scenario possible. To act as a SP in both developed applications, we used a virtual server located in London (average Round-Trip Time (RTT) of 72 ms). This server was provided by DigitalOcean\(^1\) and it has 2 GB of memory, 40 GB of disk and uses an Ubuntu 16.04.3. For the client, we used two Macbook Pro running macOS Sierra with 16 GB of memory, 500 GB of disk and Chrome browser to load the applications. All the developed tests are JavaScript scripts that run with the applications. We tried to execute them over the same conditions to avoid potential network issues and their results are always presented using the Cumulative Distribution Function (CDF).

7.2.1 Performance Tests

Considering the requirements of the developed applications, we defined three types of performance tests: deployment, user search and message delivery tests.

The aim of the deployment tests was to measure the deployment time of each application in the client browser. These tests are particularly relevant because the application supported by the reTHINK framework has some elements that are deployed at execution time (namely Runtime and Hyperties), which can have a huge impact on its deployment time. The Table 7.6 presents an overview about the performed deployment tests.

For the application supported by the reTHINK framework we did three different tests, see Figure 7.1. Note that the scale is different in each figure. The first test was the Runtime installation, which consisted in the measurement of the Runtime installation time, throughout 100 executions. Then, we proceeded with the Hyperties (Group Chat Manager and Connector) deployment test. In this second test, we measured the deployment time of each Hyperty, consecutively and concurrently, in 100 executions. And finally, we finish the deployment tests with the web page loading test (e.g. scripts and UI).

On the other hand, the application supported by the State of the Art has no elements deployed at execution time, with the exception of the web page shown to the client. For this reason, with this application, we only used the web page loading test (see Figure 7.2).

As expected and as a consequence of the installation at execution time, the use of the reTHINK

\(^1\)https://www.digitalocean.com/ , last accessed August 22th, 2017
Table 7.6: Deployment tests overview

<table>
<thead>
<tr>
<th>Applications</th>
<th>Test</th>
<th>Minimum Time</th>
<th>Average Time</th>
<th>Maximum Time</th>
<th>Total Time Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video conference application supported by the reTHINK framework</td>
<td>Runtime Installation</td>
<td>701 ms</td>
<td>857 ms</td>
<td>2135 ms</td>
<td>2173 ms</td>
</tr>
<tr>
<td></td>
<td>Hyperties Deployment</td>
<td>671 ms</td>
<td>934 ms</td>
<td>3000 ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(concurrently)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Web Page Loading</td>
<td>290 ms</td>
<td>382 ms</td>
<td>1358 ms</td>
<td></td>
</tr>
<tr>
<td>Video conference application supported by the State of the Art</td>
<td>Web Page Loading</td>
<td>260 ms</td>
<td>343 ms</td>
<td>1729 ms</td>
<td>343 ms</td>
</tr>
</tbody>
</table>

Figure 7.1: Deployment tests of the application supported by the reTHINK framework

framework has a huge impact in the deployment time (see Figure 7.3). The deployment of the application supported by the reTHINK framework takes on average more 1830 ms and, in the worst case 4764 ms more. We think that this time is relevant enough to bother users and developers, and we will try to confirm this in the last part of this evaluation. To improve this deployment time, one possible solution would be to create a lighter version of the Runtime, excluding some internal components.

The user search tests are designed to test the efficiency of the discovery of users using the discovery system provided by each developed application. On one side, we have the discovery system provided by the reTHINK framework that allows a local (using only the Domain Registry) or a global discovery (using the Domain Registry, Global Registry and Discovery Service) of users across several SP domains. And
Figure 7.2: Web page loading test of the application supported by the State of the Art

Figure 7.3: Deployment test of both developed applications

on the other side, we have the discovery system provided by the application supported by the State of the Art that is limited to the active database on the SP. The Table 7.7 presents an overview about the performed user search tests.

Table 7.7: User search tests overview

<table>
<thead>
<tr>
<th>Applications</th>
<th>Concurrent Requests</th>
<th>Minimum Time</th>
<th>Average Time</th>
<th>Maximum Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video conference application supported by the reTHINK framework</td>
<td>1</td>
<td>63 ms</td>
<td>73 ms</td>
<td>97 ms</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>66 ms</td>
<td>85 ms</td>
<td>181 ms</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>69 ms</td>
<td>97 ms</td>
<td>200 ms</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>120 ms</td>
<td>190 ms</td>
<td>345 ms</td>
</tr>
<tr>
<td>Video conference application supported by the State of the Art</td>
<td>1</td>
<td>84 ms</td>
<td>99 ms</td>
<td>162 ms</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>95 ms</td>
<td>180 ms</td>
<td>247 ms</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>162 ms</td>
<td>436 ms</td>
<td>510 ms</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>321 ms</td>
<td>882 ms</td>
<td>1000 ms</td>
</tr>
</tbody>
</table>

Although, for comparison purposes only the local discovery (see Figure 7.4a) provided by the reTHINK framework is relevant, we also tested the global discovery (see Figure C.1). For the global discovery tests, we collected the time necessary to complete a query to the Discovery Service, Global Registry, Domain Registry and the total query time. On the other hand, for the local discovery tests, we only gathered the time necessary to complete the query that, in this case, only interacts with the Domain
Registry. In addition, we varied the number of concurrent requests, using 1, 2, 5, and 10, for every 100 executions. We used a utility module called Async\(^2\) to trigger the concurrent requests. Additionally, we used the same test methodology for the discovery system provided by the application supported by the State of the Art (see Figure 7.4b).

![Figure 7.4: User search tests](image)

Taking into account Table 7.7 and Figure 7.4 (note that the scale is different in each figure), we can conclude that the local discovery provided by the reTHINK framework has, in all cases, a better performance. In addition, the discovery system provided by the reTHINK framework provides higher security in the communication between components, by using HTTPS and a Mutual Authentication protocol.

However, by analyzing Figure C.1, we can see that the global discovery provided by the reTHINK framework has a worse performance than the local discovery and the discovery system provided by the application supported by the State of the Art. Our opinion is that the components used in the global discovery are not very well integrated in order to provide a better efficiency, specially the Discovery Service because it is the component that introduces the higher delay.

To complete the performance tests, we designed the message delivery tests. The aim of these final tests was to measure the time necessary to deliver a message to another client using both developed applications. To exchange messages, the application supported by the reTHINK framework takes advantage of the Data Object mechanism, whereas the application supported by the State of the Art uses the Data Connection object to send and receive P2P arbitrary data. In these tests, we fired 100 messages, one every 50 milliseconds, in both developed applications (see Figure 7.5).

Taking into account Table 7.8, the exchange of messages is faster in the application supported by the State of the Art. These results were already expected because the Data Object mechanism involves a bigger message flow and events propagation between components, whereas in the application supported by the State of the Art, the data is sent directly to client browser via P2P.

\(^2\)https://caolan.github.io/async/, last accessed October 17th, 2017
Table 7.8: Message delivery tests overview

<table>
<thead>
<tr>
<th>Applications</th>
<th>Minimum Time</th>
<th>Average Time</th>
<th>Maximum Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video conference application supported by the reTHINK framework</td>
<td>67 ms</td>
<td>98 ms</td>
<td>206 ms</td>
</tr>
<tr>
<td>Video conference application supported by the State of the Art</td>
<td>40 ms</td>
<td>83 ms</td>
<td>125 ms</td>
</tr>
</tbody>
</table>

In terms of security, both ensure the security of the messages in transit, however, the use of the reTHINK framework provides the Mutual Authentication protocol that ensures the authentication between clients. Additionally, the Data Object mechanism does not have to handle the NAT and P2P connection issues, which could influence a lot the results of the application supported by the State of the Art. At the end, we think that the difference between the achieved results is not relevant enough to get noticed by users and developers, as both times are low when compared to the time it takes a human to type in a message.

7.2.2 Portability Tests

For the portability tests, we tried both developed applications in different browsers. We choose to use the Google Chrome 56.0.2 and the Mozilla Firefox 55.0.2 because both should support the WebRTC technology.

The aim of these portability tests was to test the behavior of both developed applications in different environments. In particular, we want to see if the reTHINK framework could be used in different browsers without any issues. With this in mind, we tried different application requirements such as authentication of users, chat and video calls in those browsers (see Table 7.9).

Taking into account Table 7.9, we can conclude that the application supported by the reTHINK framework worked better than the application supported by the State of the Art using different browsers. However, we found some bugs in both developed applications that are related with the used technologies.

With the application supported by the State of the Art, we were unable to start video calls using Firefox. This bug is directly related with the PeerJS framework. In fact, there exists an open issue on its GitHub for this bug, but there is no suggestion to solve it.
Table 7.9: Portability tests overview

<table>
<thead>
<tr>
<th>Applications</th>
<th>Deployment</th>
<th>User Authentication</th>
<th>Creation of Chat Rooms</th>
<th>Exchange of messages</th>
<th>Establishment of Video Calls</th>
<th>Browsers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video conference application supported by the reTHINK framework</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Google Chrome, Mozilla Firefox</td>
</tr>
<tr>
<td>Video conference application supported by the State of the Art</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Google Chrome, Mozilla Firefox</td>
</tr>
</tbody>
</table>

On the other hand, with the application supported by the reTHINK framework, we were unable to use two simultaneous Firefox windows (one normal and another anonymous) to test the application. This problem seemed to be related with the version of Dexie\(^3\) (used in reTHINK framework) with this Mozilla Firefox version. To continue testing this application using a single computer, we had to use one Firefox window and another Chrome window.

Most of these bugs appear because this type of technology is constantly changing, especially the browsers. A framework for web applications development needs to be aware of these changes so it can keep up with this evolution. Otherwise, it the appearance of such problems is expected over time.

### 7.3 Users and Developers Evaluation

The last part and probably the most important part of this evaluation is the users’ and developers’ evaluation. At this point, the opinions of users and developers are extremely important for the reTHINK project. The concepts introduced by the reTHINK framework are implemented, also its main goals were achieved. However, if users find it interesting, easy to use and whether that brings benefits in development are two completely different questions. To hopefully answer these questions, we divided this last part into two different evaluations.

The first evaluation consisted on some tests with users using the developed applications. The aim with this evaluation was to gather some initial feedback about the reTHINK framework through the comparison of both applications, in terms of UI and features. Additionally, we wanted to check if the differences between the efficiency of the features of both applications are relevant or not to the users.

The last evaluation involves some tests with developers with the aim of collecting useful feedback. These tests were made on some programmed events, where experienced developers had the opportunity to use the reTHINK framework to develop their own web applications. In the end, we conducted an analysis over the answers, allowing us to reach certain conclusions about the reTHINK framework, in particular its advantages and disadvantages.

\(^3\)http://dexie.org/, last accessed August 24th, 2017
7.3.1 Tests with Users

To be able to get a relevant feedback, we gathered 10 users (8 males and 2 females) to try both developed applications. Only 20% of the users have a background related with computing science, which means that most of them were not familiar with application development and programming. In addition, we used the same test scenario of the applications evaluation presented on Section 7.2.

Although, this first evaluation consisted in using each application, testing its UI and features, only the authentication form and identities management features (e.g. change the authenticated account and application logout) are different. For this reason, only these aspects were relevant for comparison purposes between the UI of the applications.

In order to collect useful feedback, we asked the users to load and authenticate themselves in both applications. In addition, we gave them complete freedom to use the chat room and video calls features. After having used both applications, we presented a survey to gather the desired user feedback.

In the first couple of questions, we tried to understand if the difference of deployment time between the applications was relevant enough to be noticed by the users. So, we started by asking which of the applications took longer to load (see Figure 7.6) and if this loading time is relevant enough or not (see Figure 7.7). Considering the answers to these questions, 80% of the users selected the application supported by the reTHINK framework as the application that took longer to load. In addition, the same 80% of the users stated that this loading time was relevant enough.

Figure 7.6: Longest loading time  
Figure 7.7: Relevance of the loading time

These data allow to validate what we already stated in Section 7.2.1 on the deployment tests. With this in mind, we think that it's important to improve the deployment of the reTHINK components.

In the next two questions, we wanted to collect the user's opinion about the authentication form provided by the reTHINK framework. We asked each user what was the authentication form that they liked the most (see Figure 7.8) and if they would rather have their own custom authentication form (see Figure 7.9). In addition, we gave the opportunity to the users to write some relevant comments about the authentication form provided by the application supported by the reTHINK framework.

The answers were clear: most users did not appreciate the authentication form provided by the reTHINK framework and they would rather have their own custom authentication form. Among all the gathered comments, we think the most relevant are: “the authentication form provided by the application supported by the reTHINK framework has a low usability degree” and “the UI of the authentication form provided by the application supported by the reTHINK framework is a bit confusing”. With these answers and comments in mind, we conclude that the authentication form should be a decision of the developer, something optional to allow its customization. The reTHINK framework should allow the development of
the authentication form or the customization of the one provided. At the same time, it should ensure that the authentication works properly.

![Figure 7.8: Preferred authentication form](image)

![Figure 7.9: Develop a custom authentication form](image)

The last question related with the authentication of users was if they managed to use the identity management features in both applications and to justify the answer. 90% of the users stated that they could not use the identity management features of the application supported by the reTHINK framework because when they tried to change the authenticated account, nothing happened. We faced this problem when we were developing this application (Section 5.2.1) and, consequently we reported this problem to the reTHINK development team.

The final part of this survey was focused on the user search features and the exchange of messages with both developed applications. The aim of this part was to see if the users noticed the difference of efficiency between applications. First, we asked what was the application that took longer to search users to start the communication (chat or video) and then, which application provides a faster messaging service (see Figures 7.10 and 7.11). Considering the variety of the answers to these questions, we conclude that the difference of efficiency of both features is not relevant enough to the users.

![Figure 7.10: Fastest user search](image)

![Figure 7.11: Fastest messaging service](image)

Although, we recognize that with 10 users, the amount of gathered feedback was limited, we experienced difficulty finding more users because these tests were done in a holiday season. Despite this limitation, this survey was an important part of the performed evaluation as we could validate all the statements and conclusions that were presented in the previous evaluation and implementation sections.

Nevertheless, we still needed to do an evaluation to gather the developer's opinion about the ease of developing applications with the reTHINK framework, testing its benefits and the cost in terms of complexity.
7.3.2 Tests with Developers

On 27th of May, INESC-ID together with AlticeLabs organized a developer competition, which took place in the Techtris House, a startup incubator and home to the WebSummit offices in Lisbon. The goals of this competition were to promote the reTHINK framework and gather the intended developer feedback. The competition had developers, who were completely unfamiliar with the reTHINK framework, complete a set of 5 challenges:

1. Create a simple web application to make use of an Hyperty and its features (i.e. generation of unique codes);

2. Change the previous developed application into a chat application that uses two different Hyperties (chat Hyperty and code generator Hyperty);

3. Change slightly the code generator Hyperty functionality using the Hyperty Toolkit. In addition, test these changes with the previously developed application;

4. Develop a video conference application with well-defined functional requirements using Hyperties already available or new ones;

5. Create a new Hyperty from scratch; Each developer was free to design the Hyperty as well as the web application that uses it.

The competition had 12 participants, grouped into 8 teams of at most two elements. Most participants were male, but a female attended. Participant’s ages ranged from 20 to 70 years. The conclusions were reached using one survey that was used to evaluate different aspects of the reTHINK framework. This survey was divided into seven sections. One section for team characterization and background, five sections corresponding to each of the challenges and a last section for overall evaluation. The collected feedback from this survey will be used to identify and improve on the weaker parts of the framework, making the system more usable. It will also allow us to evaluate how well understood are the most important concepts, so that we may improve the documentation and training material in the future.

From the first section, we noticed that all participants had a background related with computing science and, consequently with programming. However, 67% of the participants had a minimal knowledge about web application development, where only 58% of them had already an experience with frameworks for web application development. This means that most of the gathered feedback is based on the experience of the participants. And in addition, it is also interesting to see the answers of those who never did such a thing.

After that, we collected the answers of the survey related with the first challenge. The aim of this first challenge was to provide a first contact with the reTHINK framework and its concepts. All participants had to install the Runtime and load the required Hyperty from the provided SP (1st task). Then, each developed application had to use the features of the deployed Hyperty (2nd task).
Firstly, we started by analyzing how easy was to include Hyperties in the application (see Figure 7.12) and how much knowledge about the reTHINK framework do they think it was required (see Figure 7.13). Considering the answers to these questions, we see that most participants think that the difficulty on loading Hyperties is normal and that it is not necessary to acquire a good amount of knowledge about the reTHINK framework, which is very good. However, there are relevant comments that should be considered such as “Without the tutorial and further instructions, we would have taken a lot more time”; “It was necessary a good knowledge because is lacking some documentation”; “you must to have minimal understanding of the framework (conceptually) to be able to understand how to use it”.

After that, we analyze how much time they spent on each task and reading the documentation. Our estimation for the 1st task was 45 minutes and for the 2nd task was 30 minutes. Surprisingly, the average time of the participants for the 1st task was 48 minutes and for the 2nd task was 44 minutes. We consider these numbers very good for someone who never tried the reTHINK framework. In addition, 83% of the participants stated that spent less than 30 minutes reading the documentation.

To finish this first part of the survey, we asked the participants to classify their experience with the reTHINK framework in this first challenge and to write the limitations found. 50% of the participants classified their experience as good, however there are limitations that should be considered:

- “It would be useful (at least for developers) to be able to use anonymous identities”;
- “Excessive Logging”;
- “Authentication form is not very usable”;
- “It should be provided a better documentation and good tutorials to get people started”.

Even with the limitations and disadvantages found, each team completed with success this first challenge and we were able to collect very useful feedback about reTHINK and its features. There were positive aspects such as even with minimal knowledge about web application development, all participants were able to use reTHINK to complete this challenge and all participants considered reTHINK easy to use.

The second challenge was basically to load two different Hyperties into a chat application, with a much more complex UI. Each team had to load these Hyperties (1st task), develop all the UI to deal with their features (2nd task) and finally, combine its features to reach to a certain result (3rd task).
Firstly, we analyze how easy was to load more than one Hyperty (see Figure 7.14) and how much knowledge about the reTHINK framework do they think it was required (see Figure 7.15). Considering the answers to these questions, we can conclude that the participants recognized a higher difficulty when loading multiple Hyperties and higher necessity to acquire more knowledge about the reTHINK framework. In addition, most of them said that the documentation sometimes is not entirely clear, what makes things more complicated.

After that, we collected the team's development time in each task. Our estimation for the 1st and 3rd tasks was 15 minutes and for the 2nd task was 150 minutes. The average time of the teams for the 1st task was 19 minutes, for the 2nd task was 115 minutes and for the last task was 26 minutes. As we can see, the teams finished the 2nd task earlier than expected, which is outstanding. Even though, 50% of the participants have spent more than 30 minutes reading the documentation. At this point, we noticed that the teams were enjoying a lot using the reTHINK framework to develop this type of applications. The only limitation that some participants found was the inability of using two Firefox windows to test the application. This problem was also detected in Section 7.2.2 and it should be fixed in a near future.

To finish this second part of the survey, we asked to all teams if they would use the reTHINK framework again to develop another chat application and to justify their answer. We were happy to know that 100% of the participants would use it again and we received a lot positive comments:

- “With our limited experience, it was quite easy and intuitive to implement these functionalities using the reTHINK framework”;
- “Even though it was hard to start, with some practice the framework would be great to make new projects in the future”;
- “After realizing the framework operation was relatively simple to use compared to the traditional method”.

Although, only four teams managed to complete this second challenge with success, we collected a lot useful feedback. Most of it was very positive feedback. Teams enjoyed developing chat applications taking advantage of the Data Object mechanism provided by the reTHINK framework. In addition, most of them recognized that with the available Hyperties, the development is much simple and faster and, they would be happy to use it again in a near future.

![Figure 7.14: Multiple Hyperties.](image)
![Figure 7.15: Knowledge to combine Hyperties.](image)
The next challenge consisted in using the Hyperty Toolkit provided by the reTHINK framework to make a simple change on some Hyperty functionality. The goal of this third challenge was to give the opportunity to the teams to try the Hyperty Toolkit (1st task) and to do their first development related with Hyperties (2nd task). In addition, the teams had to change their applications to use the updated Hyperty from the Hyperty Toolkit and not from the SP.

In the beginning, we collected some feedback about how easy was to setup the reTHINK development environment (see Figure 7.16) and how much knowledge about the framework do they think it was required. Considering the answers to these questions, we can see that most participants considered easy to setup the reTHINK development environment. Additionally, most of them said that they had no problems in this first task, requiring very little knowledge about the framework and, 50% of the participants stated that spent less than 10 minutes reading the documentation.

Thereafter, we analyzed the teams development time in each task and how easy it was for them to change the Hyperty functionality (see Figure 7.17). Our estimation for the 1st task was 35 minutes and for the 2nd task was 40 minutes. The average time of the teams for the 1st and 2nd tasks was 51 minutes. These numbers suggested that changing the Hyperty functionality could be something difficult for developers. However, the Figure 7.17 proves the opposite. Most participants considered easy to change the Hyperty functionality. To clarify this, we asked the participants why they took so long to complete these two tasks. In their opinion, they took so long because testing and debugging with the reTHINK framework involves a lot of steps and therefore, a lot of time.

Only two teams were able to complete this third challenge, which was very good because we estimated that reach the 4th challenge in only one day would be extremely difficult for the participants. Nevertheless, we gathered a very positive feedback in this third challenge. Most participants recognized that use the Hyperty Toolkit and therefore, change the Hyperty functionality was very easy.

![Figure 7.16: Setup the reTHINK environment.](image1)

![Figure 7.17: Change the Hyperty functionality.](image2)

The fourth challenge presented at the event was to develop a video conference application. Each team had to develop a web application that had to cover some well-defined functional requirements, such as exchange of messages between users and video calls. This was an open challenge in which the participants could choose Hyperties already available or develop their own Hyperties. Unfortunately, neither of the teams that made it to this challenge, managed to finish this fourth challenge in time. Consequently, we were not able to gather relevant feedback.

The last challenge presented to the participants was to develop a new Hyperty from scratch. Each
team was free to design the Hyperty as well as the web application that uses it. However, no team reached this challenge, so we could not collect answers.

To finish the event, one last part of the survey was presented to all teams. The aim of this final part was to receive an overall evaluation about the reTHINK framework, its benefits and features. We received a very positive feedback to all questions.

In the beginning, we asked to all participants what features of the reTHINK framework look more promising. The most relevant answers were:

- “The fast chat rooms creation”;
- “It is easy to use in projects. Does not require much to run in a script”;
- “We think the most important feature of the reTHINK framework is the possibility of completely different and independent developers, developing compatible applications without needing to standardize protocols”;
- “The best thing about reTHINK is how simple it is to take the available Hyperties and build an application out of them”.

Then, we wanted to know the participant’s opinion about what are the main benefits of using Hyperties. The most relevant answers were:

- “Allows a faster development”;
- “It is very easy to use”;
- “Hyperties shine when developing complex applications that need to interact with different domains. It simplifies the process of managing connections, which greatly improves the developer experience”.

At the end, we asked to all participants what features would they like to see in the reTHINK framework in a near future. Most of them agreed that they have to explore better the framework, but they would enjoy to see some built-in integrations with front-end frameworks and more available Hyperties with different features.

This event was a success, allowing us to collect very useful feedback. This feedback is extremely important for the reTHINK project because it will allow to improve the framework in the future, making it more developer friendly and usable. Considering all the collected feedback, the following conclusions are highlighted:

- Participants found Hyperties easily to integrate into applications. However, more examples are required;
- All participants considered the reTHINK framework easy to use;
- The reTHINK library logging needs improvements because it is too verbose;
• Documentation needs to be improved, more structured, generalized examples and demos;
• All participants expressed the intention to use the reTHINK framework again in future projects;
• It would be great if the reTHINK framework was supported by most browsers;
• The authentication form needs improvements in terms of UI;
• The Hyperty Toolkit was appreciated by participants;
• It would be great if reTHINK had built-in integration with front-end frameworks and examples.

7.4 Overall Analysis and Recommendations

After analyzing this evaluation as a whole, we think that the ultimate goal of this thesis was achieved. We were able to evaluate the reTHINK framework from different perspectives, and, whenever possible, compare it with other technologies. Additionally, we tested its advantages and disadvantages through the application development using several evaluation metrics.

This evaluation was important for the reTHINK project because it allowed to identify the necessary improvements in order to make the system more usable and developer friendly in the future. Besides that, we gathered a good amount of user and developer feedback, which at this point is very important for the project. In fact, some parts of this evaluation were already used in the internal documents of the reTHINK project.

Taking into account all the conclusions made at each part of this evaluation, we can provide a few recommendations to the reTHINK project. The most important recommendation and probably, the most urgent one, is to improve their documentation. At this point, the reTHINK framework is ready to be used by developers, however, it is extremely difficult to understand its concepts and features just by reading its documentation. At the moment, the reTHINK development team is improving the documentation available in each GitHub repository. However, we think it is necessary to go further. In our view, the reTHINK project should gather a team just to improve the documentation through the development of a website to expose better the information to be more appealing and easier to understand. Additionally, if the developers are responsible for the Hyperty development, it is important to provide a page only about the APIs that they should use, with concrete and generalized examples (e.g. how to create a Data Object).

The second recommendation is related to the size of the Runtime that each application must use. We think that its size is too big: not only does it affect the deployment time of the applications, but it also could cause the loss of interest of some developers. There are several components in the Runtime that are not used, so this problem could be quickly solved by creating a lighter version without these components.

The next and final recommendation is related with the global discovery introduced by the reTHINK project. This discovery process is essential to allow the interoperability: without it every user must know the SP domain of the other users that they want to communicate, which can be bad for developers. Our
suggestion to improved the global discovery is to upload the information of the Discovery Service to the Global Registry constantly. In addition, we think that the Global Registry should be better integrated with other components, taking more advantage of its features. This recommendation changes a little the architecture of the reTHINK framework, but the global discovery would be much more efficient and appreciated by developers and users, even with its limitations, i.e. in terms of efficiency.

To conclude, we think that it is also important to highlight the positive aspects collected with this evaluation. Most of the users and developers recognized the potential of the reTHINK framework and its uniqueness. They also enjoyed using the reTHINK framework to develop their applications and have shown interest in using it in the future. This shows that the reTHINK framework can be very well received by developers and it has a great margin of progression with a few improvements and integrations.

7.5 Chapter Summary

In this chapter, we described in detail the evaluation carried out in this work. To evaluate the reTHINK framework from different perspectives, we divided this evaluation into three parts. In the first part, we evaluated the documentation, programming effort, development time and, ease of installation and deployment of the reTHINK framework when compared with the traditional technologies. Then, the applications evaluation allowed us to do some performance and portability tests that are directly related with the used technologies. And finally, the last part of this evaluation allowed us to gather some users’ feedback related with the developed applications and some developers’ feedback about the reTHINK framework. All these evaluations allowed us to determine the necessary improvements and corrections that the reTHINK development team needs to do in order to make the system more usable in the future.

At the end of this chapter, we provided an overall analysis and a few recommendations that should be considered.
Chapter 8

Conclusions

This document describes an evaluation methodology focused on the reTHINK project. At this point, this European project has recently finished and, consequently, it was necessary to collect an evaluation not only focused on the efficiency of its components but also on the ease of developing applications.

With this in mind, we aimed to evaluate the reTHINK project from the point of view of ease in developing applications when compared with traditional technologies, testing the benefits and the costs in terms of complexity. The achievements of this evaluation, allowed us to provide important recommendations to the reTHINK project, so it can be more usable, easy to use and efficient in the future. In fact, some parts of this evaluation were used in the final review document of the reTHINK project.

This chapter reflects on our contributions to the reTHINK project, namely what was possible to achieve and conclude. We know that this work has a high degree of subjectivity because most of the collected data derives from our programming skills, knowledge and the technologies that we choose. However, most of the companies, when develop something new, follow the same path and yet, most of them do not evaluate considering so many points of view. With this in mind and although we recognized its subjectivity, we think that we did what was necessary to be done in order to achieve the main goal of this thesis and present a solid evaluation and important recommendations to the reTHINK project.

At the end of this chapter, we discuss some possible future work.

8.1 Summary

To define the most suitable evaluation methodology for the reTHINK project and, given the focus of the project in provide a framework for web application development, we began this work by gathering a background related to web application development. The main idea with this background was to provide an overview about all the protocols, mechanisms and strategies used nowadays in different aspects of this area such as communication and authentication of users.

Afterwards, we elaborate the state of art of frameworks for web application development. We started by reflecting on the considerations that every developer needs to take into account when choosing or evaluating a framework. Then, we presented an in depth view of the reTHINK framework, its components
and features. Additionally, we provide an overview about different types of frameworks that are normally used for web application development.

We had two main goals with this chapter of the state of the art. Firstly, we wanted to compare the reTHINK framework with all the presented frameworks, highlighting the main differences and similarities between them. Those comparisons have allowed us to reach some preliminary conclusions about the reTHINK framework and its features. And then, this chapter, together with the background, facilitated the choice of the most suitable technologies for the development of the application supported by the state of the art.

Taking into account the use cases targeted by the reTHINK framework and its main features, we have successfully defined an appropriate evaluation methodology to accomplish the main goal of this work. We structured two versions of the same proposed web application. One version that tries to take advantage only of the reTHINK framework, and another version that uses a wide range of technologies selected from Chapters 2 and 3. In addition, we structured an evaluation from different perspectives to collect data about several aspects such as application development and, the impact on users and developers.

Along the implementation of both applications and the evaluation, it was possible to gather a wide range of conclusions, specially about the advantages and disadvantages of using the reTHINK framework. The implementation was also important to identify limitations related with its features.

With this in mind, we think that the main goal of this thesis was achieved with success. We managed to evaluate the reTHINK framework for web application development, testing its benefits and costs in terms of complexity. During this evaluation, we tried always to compare both developments to have a term of comparison, in order to determine if the collected data were good or not.

The huge interest of the users and developers (Section 7.3) confirmed that the reTHINK framework has a tremendous potential in this so overcrowded market. Most of them said that the reTHINK framework allows for rapid application development and recognized the value of the discover and communication features. We think that following our recommendations (Section 7.4), the reTHINK framework will be more usable, easy to use and, consequently, more well accepted and adopted by the developers community.

8.2 Future Work

While we have achieved our main goals, this work may still be improved. If the reTHINK framework goes to the market and starts advertising its concepts and features, it will be important to perform a statistical analysis over the developer’s reaction. By this reaction, we mean the number of downloads, questions about the framework and developed applications.

Additionally, it will be also important to make all the reTHINK repositories on GitHub public, so the developers can start discussions about implementing certain features using the reTHINK framework. With this, the reTHINK development team can identify potential problems and improvements that they need to do in the system. Sometimes, in these discussions, developers suggest very good solutions for
that specific issue.

When the reTHINK project reaches this step, it will be possible to collect a much larger amount of feedback from developers and users. It is important to always consider this feedback and meet the developers’ and users’ needs. With this in mind, the programming events should be repeated again with the aim of gathering more feedback. Also, it would be interesting to do these events in different countries with the collaboration of the other partners of the project.
Appendix A

Database systems

A.1 SQL vs NoSQL

Table A.1: SQL vs NoSQL Databases.

<table>
<thead>
<tr>
<th></th>
<th>SQL Databases</th>
<th>NoSQL Databases</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Types</strong></td>
<td>One type with minor variations</td>
<td>Many types including key-value stores, document databases and graph databases</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>MySQL, Postgres, Microsoft SQL Server</td>
<td>MongoDB, Cassandra, HBase, Neo4j</td>
</tr>
<tr>
<td><strong>Data Storage Model</strong></td>
<td>Individual records are stored as rows in tables, with each column storing a specific piece of data about that record. Related data is stored in separate tables, and then joined together when more complex queries are executed</td>
<td>Varies based on database type. For example, document databases do away with the table-and-row model altogether, storing all relevant data together in single document in JSON, XML, or another format</td>
</tr>
<tr>
<td><strong>Schemas</strong></td>
<td>Structure and data types are fixed in advance. To store information about a new data item, the entire database must be altered, during which time the database must be taken offline</td>
<td>Typically dynamic, with some enforcing data validation rules. Applications can add new fields on the fly, and unlike SQL table rows, dissimilar data can be stored together as necessary</td>
</tr>
<tr>
<td><strong>Scaling</strong></td>
<td>Vertically, meaning a single server must be made increasingly powerful in order to deal with increased demand. It is possible to spread SQL databases over many servers, but significant additional engineering is generally required</td>
<td>Horizontally, meaning that to add capacity, a database administrator can simply add more commodity servers or cloud instances. The database automatically spreads data across servers as necessary</td>
</tr>
<tr>
<td><strong>Data Manipulation</strong></td>
<td>Specific language using Select, Insert, and Update statements</td>
<td>Through object-oriented APIs</td>
</tr>
</tbody>
</table>
Appendix B

Developed Applications

B.1 Application supported by the reTHINK Framework

B.1.1 User Profile

![Profile Picture](Profile Picture)

---

**rethink.inesc.test**

Email: rethink.inesc.test@gmail.com

Hyperty URL: hyperty://localhost:382665c6-bc56-4398-b8e4-2f75dc64f4f

![Chat Room Invitation](Chat Room Invitation)

**Chat Room Invitation**

![Accept Reject](Accept Reject)

Figure B.1: Application user profile.

Figure B.2: Invitation to join the chat room.
B.2 Application supported by the State of the Art

B.2.1 User Profile

![Profile Picture](image)

Inesc rethink
Email: rethink.inesc.test@gmail.com
PeerJS ID: 5995c7b32821411aeb75168

Figure B.3: Application user profile.

B.2.2 Chat Room Invitation

![Chat Room Invitation](image)

Invitation
Chat Room Name: Inesc rethink
Chat URL: room://state-of-the-art/998809260
Who invited: bernardo.marques@gmail.com

Figure B.4: Invitation to join the chat room.
Appendix C

Evaluation

C.1 Applications Evaluation

Figure C.1: Global discovery tests.
Bibliography


[42] James Lewis and Martin Fowler: Microservices: A Definition Of This New Architectural Term (2014)