TABLE OF CONTENTS

1

Introduction

1.1 Document overview

Este documento tiene como objetivo analizar y describir la arquitectura propuesta para el sistema PacM en el cual se describe:

El sistema consiste en un juego multijugador basado en el popular juego de Pacman; Este juego posee una característica diferente ya que fue diseñado para ser jugado de forma colaborativa utilizando la herramientas vistas en el curso, existen dos equipos: el equipo atacante (basado en la jugabilidad del pacman) y el equipo protector (basado en la jugabilidad de los fantasmas), cada equipo compuesto por al menos dos jugadores y máximo 4 jugadores, y cuya misión es distinta, el equipo atacante comer todos los puntos en pantalla y el equipo protector evitar que esto pase, en cierta cantidad de tiempo.
La arquitectura lógica de PacM está compuesta por tres capas:

Capa de presentación, Capa lógica y Capa de persistencia:

El componente *Mom* ofrece un servicio de mensajería Stomp, el cual es usado por el componente *STOMPMessageHandler*, y el Componente *Api-JavaScipt* para la comunicación entre los distintos clientes y el servidor.
El componente *Api-rest* Ofrece un servicio rest que es usado por el Componente *Api-JavaScript* para realizar peticiones web desde el cliente al servidor.

Cada computador del cliente tiene el componente *ApiJavascript*, a la hora de ingresar a la página éste se comunica a través del protocolo http con el servidor del juego que tiene los componentes *Apirest* y *STOMMessageHandler*.
Tanto el cliente como el servidor se comunican con el *Servidor MOM* a través del protocolo TCP/IP para acceder al componente Mom de mensajería

1.2 Abbreviations and Glossary

1.2.1 Abbreviations

MOM : Message-oriented middleware
API : Application Programming Interface
STOMP : Streaming Text Oriented Messaging Protocol

1.2.2 Glossary

REST : Es un estilo de arquitectura de software para sistemas de hipermedia distribuidos
STOMP: Es un simple protocolo basado en texto, diseñado para trabajar con MOM. Proporciona un formato interoperable que permite a clientes STOMP hablar con cualquier mensaje broker que soporta el protocolo.

API: **es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.**

1.3 References

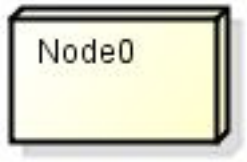
1.3.1 Project References

#	Document Identifier	Document Title
1	1	https://en.wikipedia.org/wiki/Streaming_Text_Oriented_Messaging_Protocol
2	2	https://en.wikipedia.org/wiki/Representational_state_transfer
3	3	https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones

1.3.2 Standard and regulatory References

#	Document Identifier	Document Title
[STD1]		Add your documents references. One line per document

1.4 Conventions



Nodo físico de la aplicación.

Componente lógico de la aplicación.

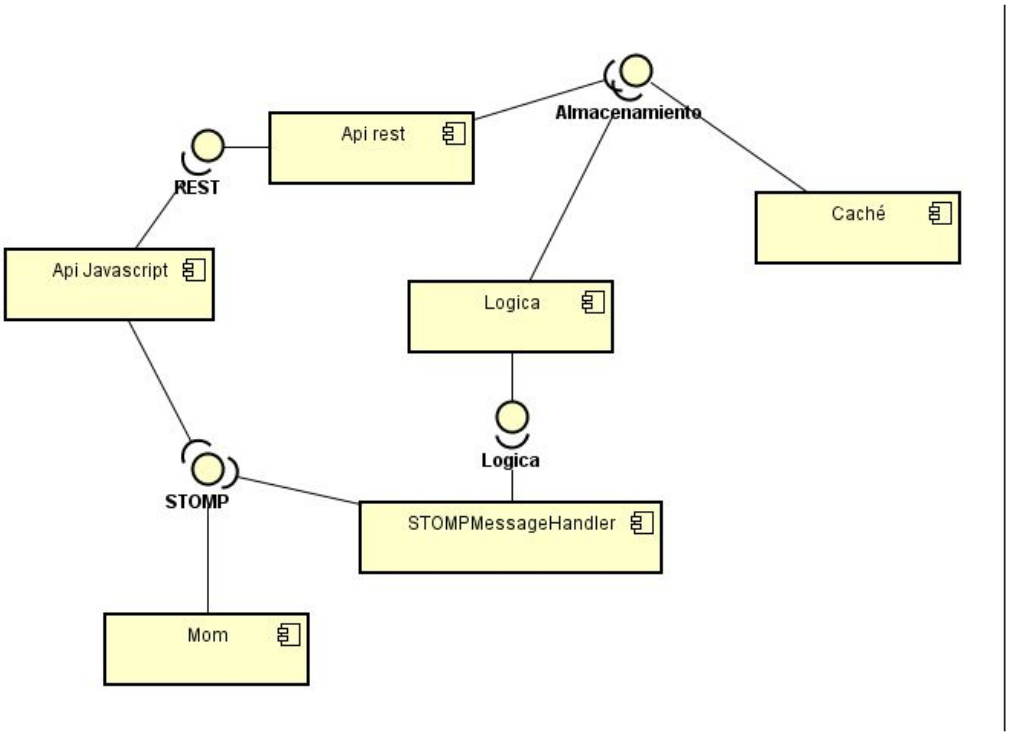
2

Architecture

2.1 Architecture overview

- Este proyecto funciona en ambientes hogareños donde cualquier persona pueda jugar Team Pam
- Cualquier persona que quiera jugar en linea nuestra versión de pacman
- Es una plataforma de juego en línea,
- Las funciones principales son: escoger el nombre de usuario, escoger en qué equipo jugar, jugar al juego y declarar el equipo ganador

2.2 Logical architecture overview



Existen actualmente 5 componentes, en la arquitectura lógica, el componente ApiJavascript representa al cliente el cual requiere servicios REST y Stomp, los cuales provienen el REST del servicio ofrecido por el componente Apirest, y el servicio STOMP el cual es ofrecido por el componente MOM, no sólo el ApiJavascript utiliza el servicio STOMP, si no también el componente STOMPMessageHandler.
El componente Apirest utiliza un servicio de almacenamiento el cual lo provee un stub.

2.2.1 Software Component 1 description

Describe the content of each top level software component in the architecture
Optional , you may not do it for 2 rationale :

1. either your software is not class C according to IEC 62366
2. or you describe each top level component in a SDD.

The description shall contain :

- **Its identification**
- The purpose of the component,
- Its interfaces with other components,
- Its network interfaces,
- The hardware resources it uses, for example : average RAM usage, peak RAM usage and peak frequency and duration, disk space for permanent data, disk space for cache data, average CPU usage, peak CPU usage and peak frequency and duration …

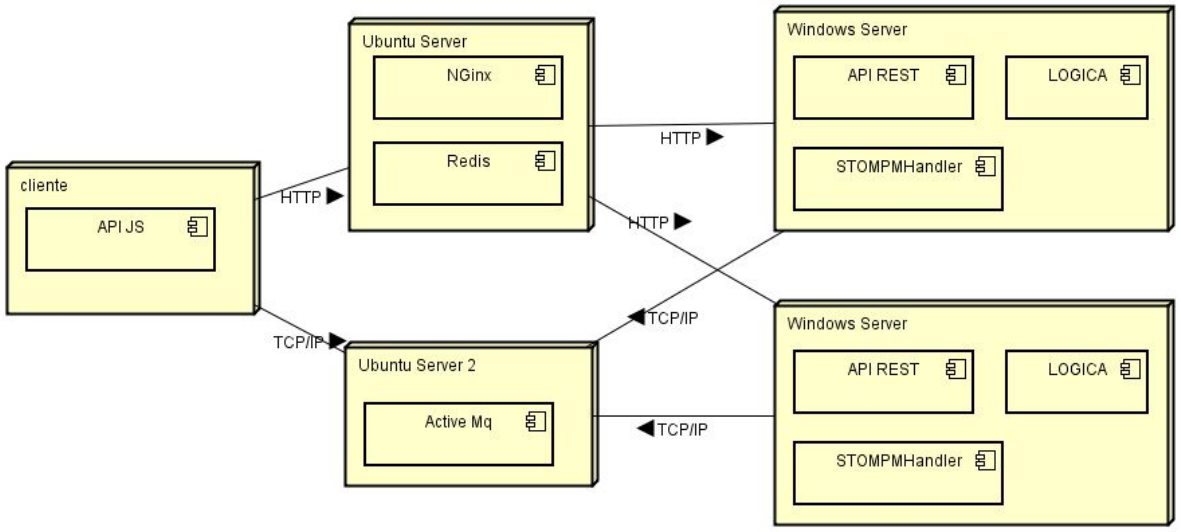
2.2.2 Software Component 2 description

Repeat the pattern for each top level component.

2.2.3 Software Component 3 description

Repeat the pattern for each top level component.

2.3 Physical architecture overview



Se tienen 3 nodos fisicos, en donde el nodo cliente no es único, es decir, muchos clientes se comunican con el servidor y el servidor MOM.
Los clientes se comunican a través del protocolo http con el servidor el cual contiene los componentes de APi rest, el stub y el StompMessageHandler.
Tanto el cliente como el servidor utilizar el servidor MOM para su comunicación a través del protocolo TCP/IP, el servidor MOM Tiene el componente MOM.

[SEI: Documenting software architectures: Chapter 5 (5.2)

2.3.1 Hardware Component 1 description

Describe the content of each hardware component in the architecture
Optional , you may not do it if your software is not class C according to IEC 62366
The description shall contain :

- Its identification
- The purpose of the component
- The software component it receives
- Its technical characteristics : type of machine, CPU, RAM, disk and so on.
- Its network hardware interfaces

2.3.2 Hardware Component 2 description

Repeat the pattern for each top level component.

2.3.3 Hardware Component 3 description

Repeat the pattern for each top level component.

2.4 Software COTS

If you use COTS (Components Off the Shelf, also named SOUP, Software Of Unknown Provenance), list them here.
For each COTS, describe :

- Its identification and version
- Its purpose
- Where it comes from : manufacturer, vendor, university …
- Wether it is maintained by a third party or not
- If this is an executable,
    - what are the hardware / sotfware resources it uses
    - Wether it is insulated in the architecture and why
- Its interfaces and data flows

Note : have a look at FDA Guidance « Off-The-Shelf Software Use in Medical Devices » to determine if you need specific or special documentation for your COTS.

3

Dynamic behaviour of architecture

The architecture was designed to answer to functional requirements.
For each main function of the system, add a description of the sequences / data flow that occur.
Use sequence diagrams, collaboration diagrams

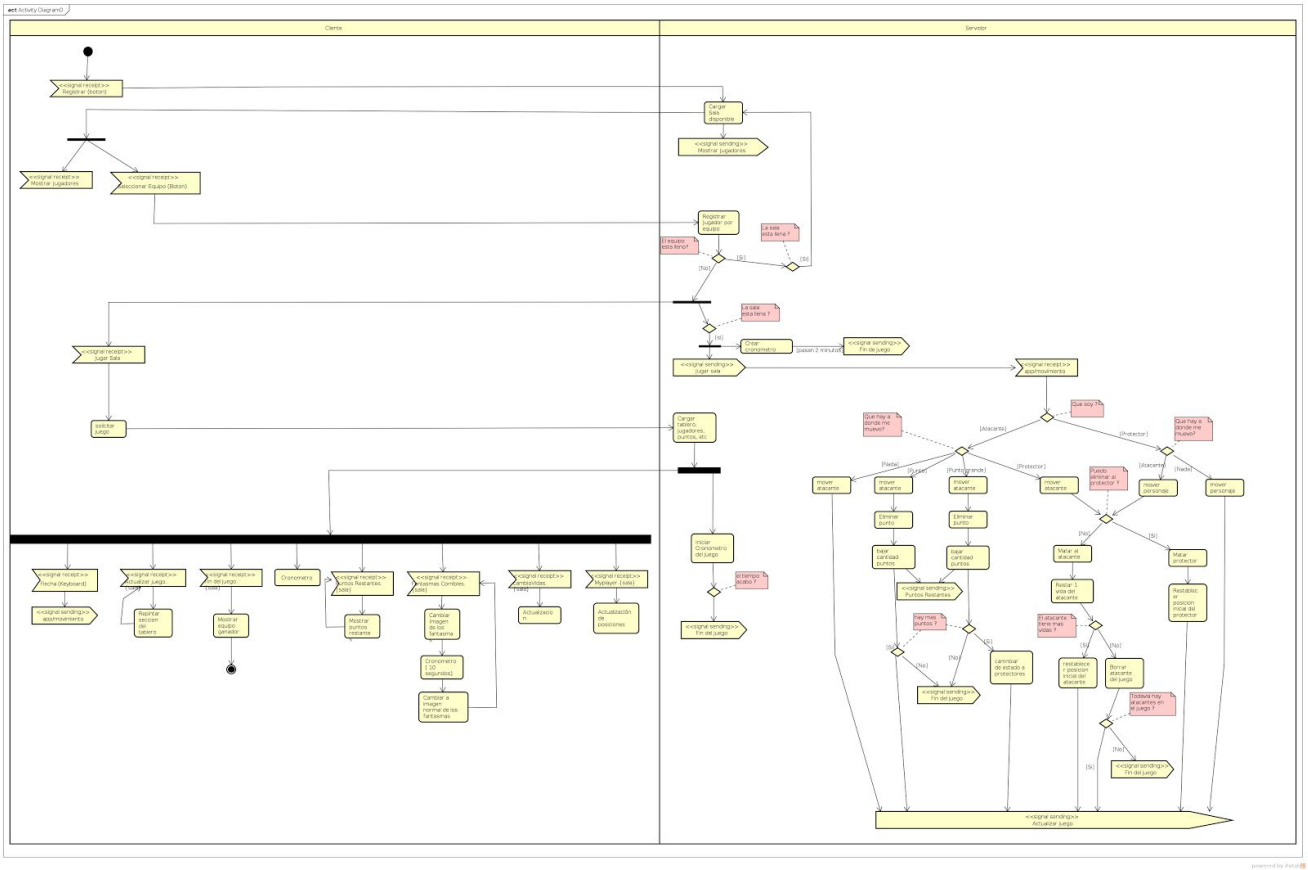
[SEI: Documenting software architectures: Chapter 8 (8.2) : Sequence Diagrams, ActivityDiagrams (for event-based behavior).

3.1 Workflow / Sequence 1

Describe here the workflow / sequence of a main function
For example, the user queries data, what happens, from his terminal to the database.

3.2 Workflow / Sequence 2

Repeat the patern for each main function of the system



4 .Justification of architecture

3.3 System architecture capabilities

Describe here the rationale of the hardware / software architecture in terms of capabilities :

- Performances (for example response time, user mobility, data storage, or any functional performance which has an impact on architecture)
- User / patient safety (see §4.3 and §4.4)
- Protection against misuse (see 4.4)
- Maintenance (cold maintenance or hot maintenance),
- Adaptability, flexibility
- Scalability, availability
- Backup and restore
- Hardware and Software security : fault tolerance, redundancy, emergency stop, recovery after crash …
- Administration,
- Monitoring, audit
- Internationalization

3.4 Network architecture capabilities

If the medical device uses/has a network, describe here the rationale of the hardware / network architecture :

- Bandwidth
- Network failures
- Loss of data
- Inconsistent data
- Insconsistent timing of data
- Cyber security (see FDA Guidance on Cyber Security of networked medical devices)

3.5 Risk analysis outputs

If the results of risk analysis have an impact on the architecture, describe here for each risk analysis output what has been done to mitigate the risk in the architecture.
Use diagrams if necessary, like architecture before risk mitigation and architecture after risk mitigation, to explain the choices.

3.6 Human factors engineering outputs

If the results of human factors analysis have an impact on the architecture, describe here for each risk human factors output what has been done to mitigate the risk in the architecture.

4

Requirements traceability

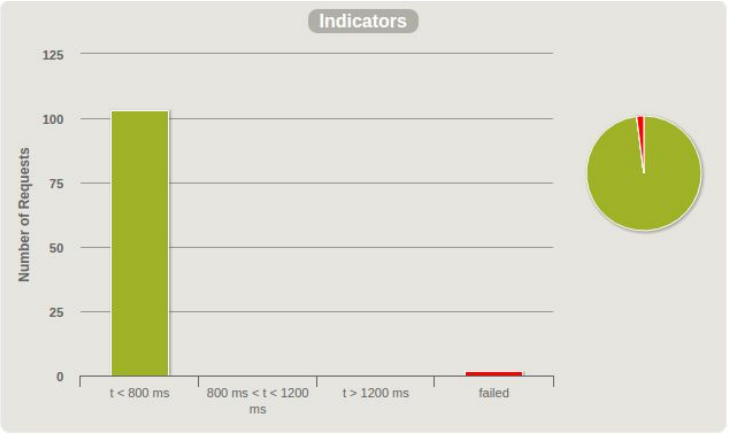
Add a table with traceability of components of this document with functional requirements.

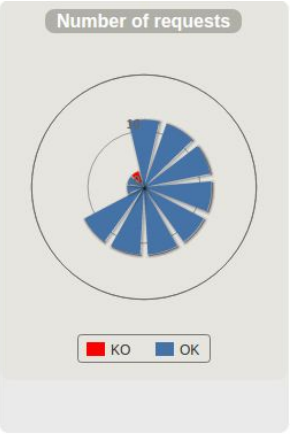
Requirement	Component	Comment
REQ-001 The device shall do foo	COMPO-001: foo maker	COMP-001 does foo. COMP-002 also does verification of foo.

This may be a difficult job. A high level function is usually handled by many components. In this case, quote only the component(s) which has(have) the major role.

6. Performance analysis

6.1 First Version



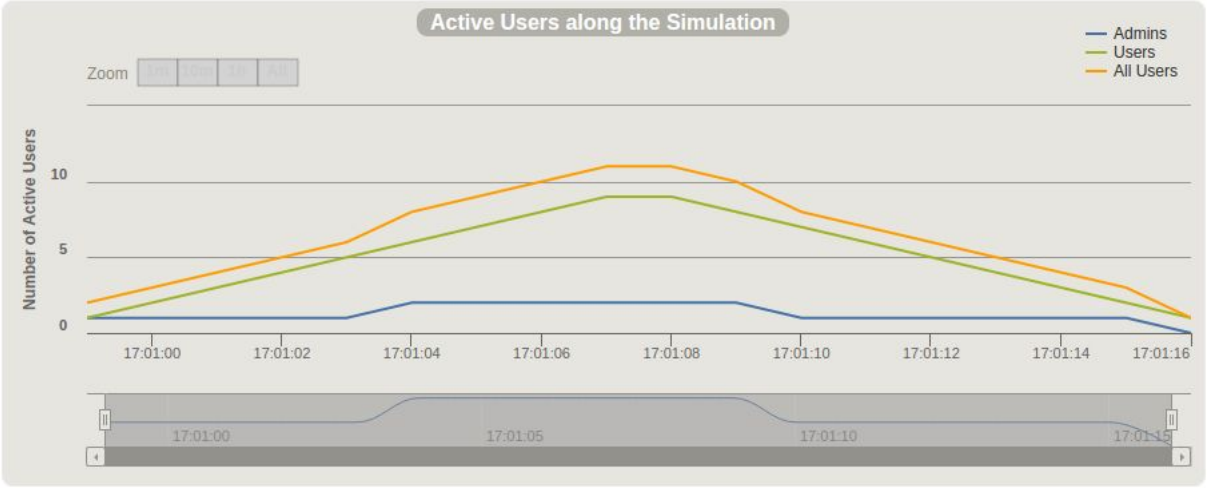


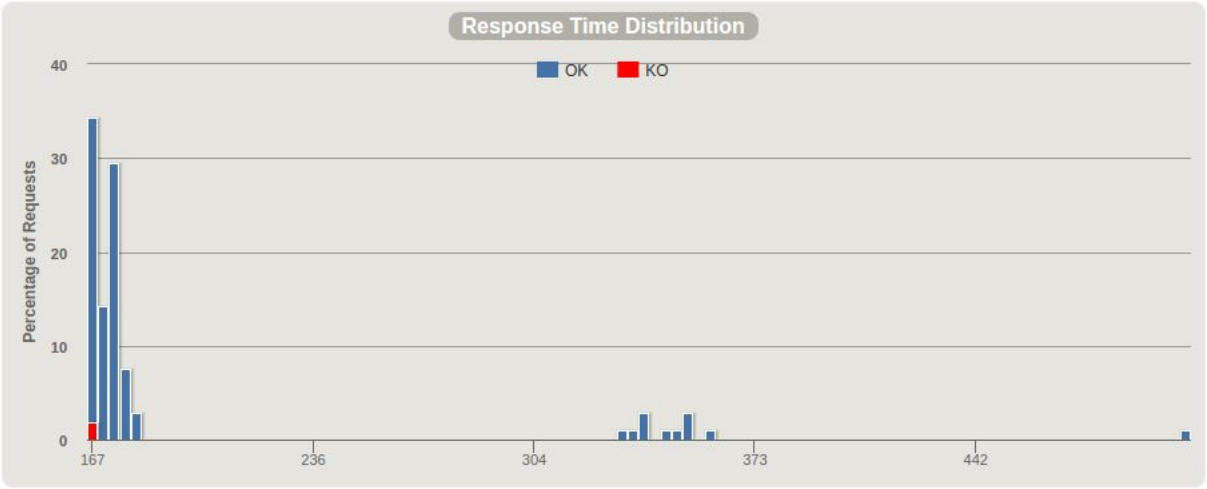
STATISTICS — Expand all groups | Collapse all groups

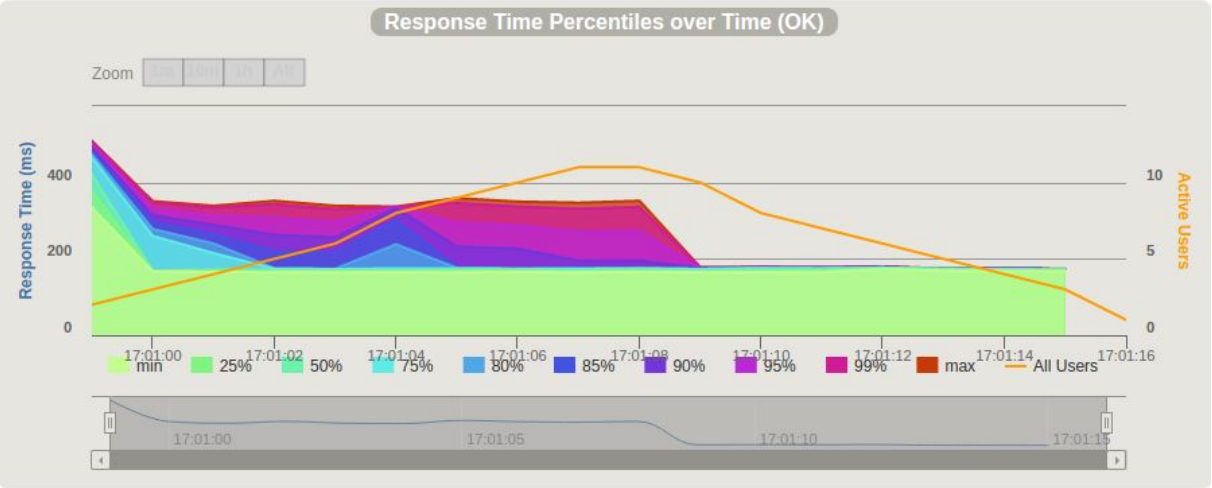
Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	105	103	2	2%	5.833	165	172	175	349	359	509	192	62
Home	12	12	0	0%	0.667	333	349	352	426	493	509	359	46
Home Redirect 1	12	12	0	0%	0.667	166	172	174	176	178	178	171	4
Search	12	12	0	0%	0.667	166	172	174	177	179	179	172	4
Select	12	12	0	0%	0.667	165	172	174	177	178	178	171	4
Page 0	12	12	0	0%	0.667	165	171	175	177	178	178	171	4
Page 1	12	12	0	0%	0.667	165	171	174	177	179	179	171	4
Page 2	12	12	0	0%	0.667	165	171	175	177	178	178	171	4
Page 3	12	12	0	0%	0.667	165	172	174	177	179	180	171	4
Form	3	3	0	0%	0.167	166	168	169	169	169	169	168	1
Post	3	3	0	0%	0.167	167	168	169	169	169	169	168	1
Post Redirect 1	3	1	2	67%	0.167	165	168	168	168	168	168	167	1

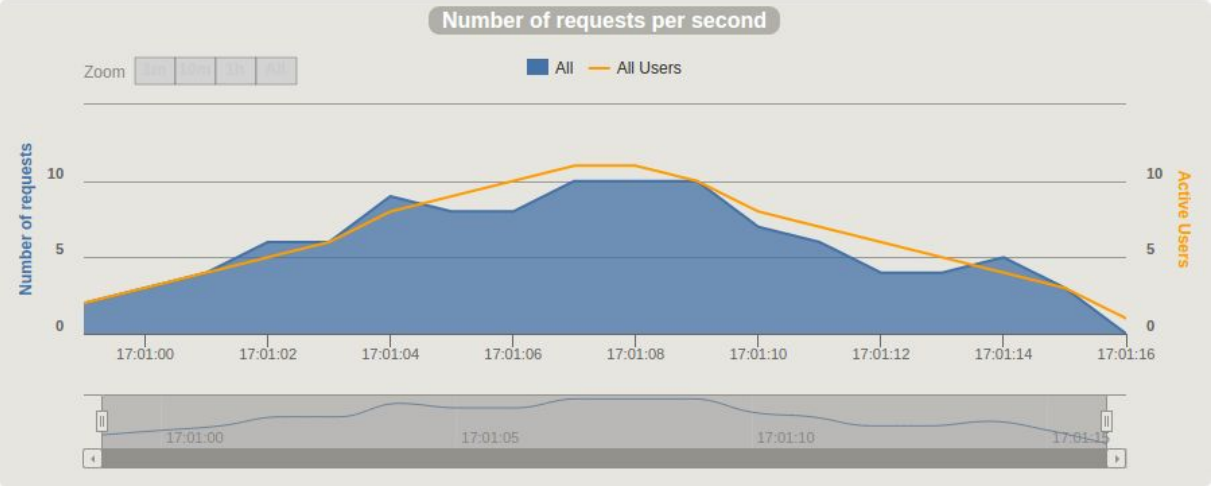
ERRORS

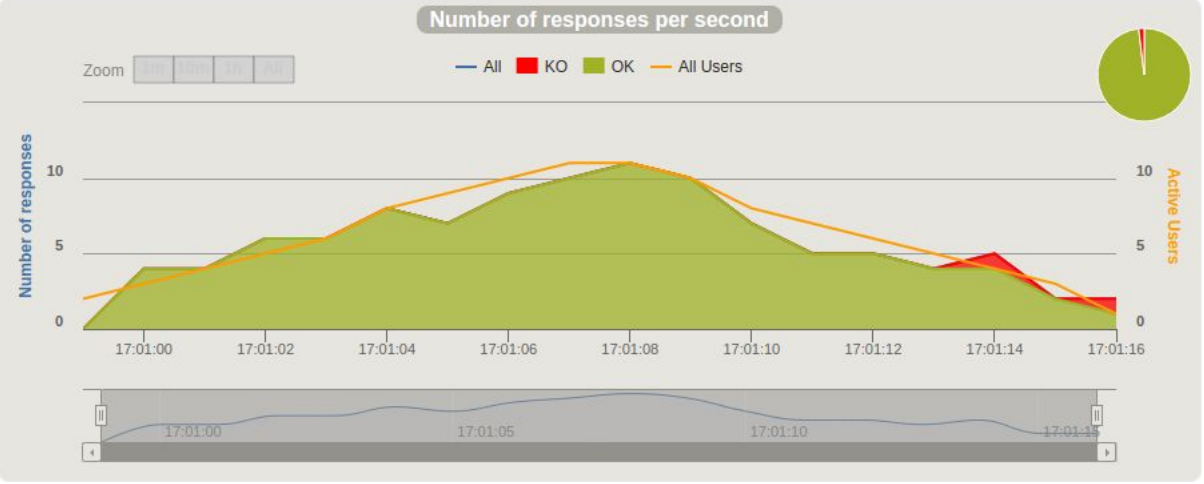
Error	Count	Percentage
status.find.is(201), but actually found 200	2	100 %



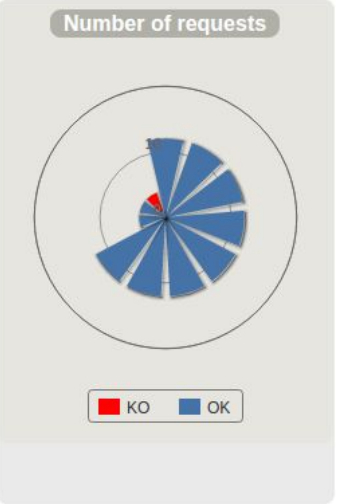








6.2 Second Version

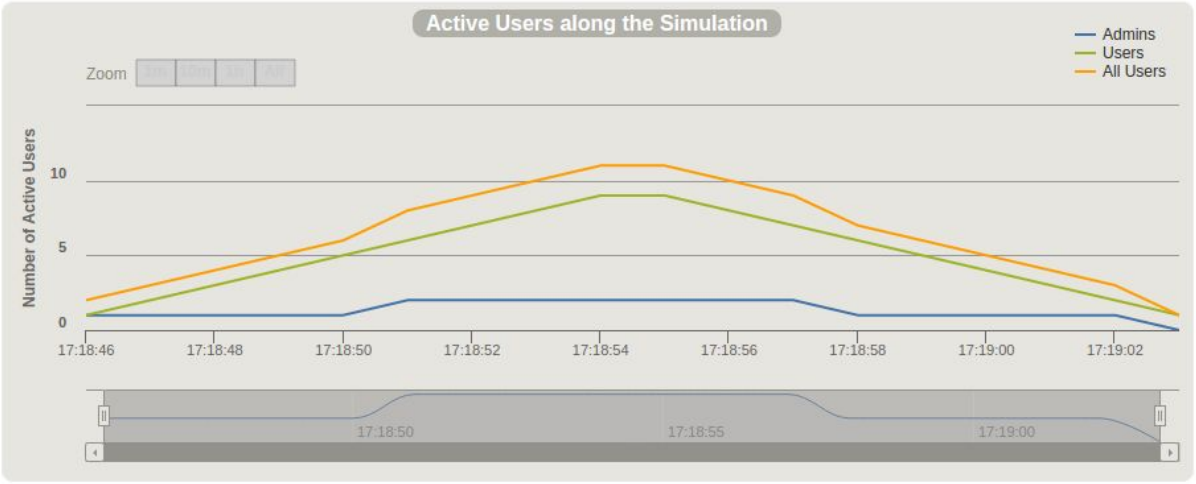


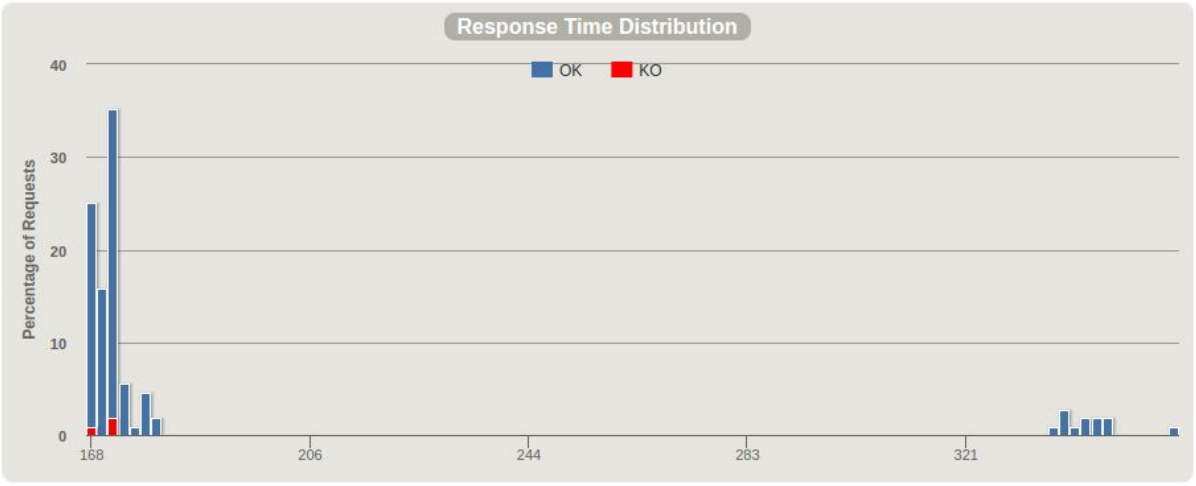
STATISTICS — Expand all groups | Collapse all groups

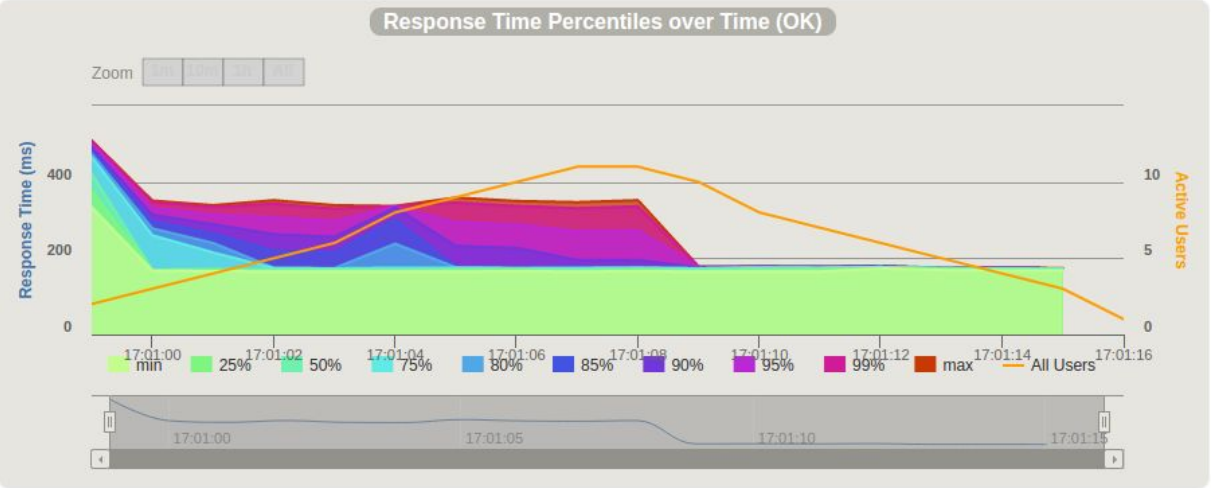
Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	108	105	3	3%	6	167	171	172	342	346	358	190	54
Home	12	12	0	0%	0.667	336	342	344	351	357	358	342	6
Home Redirect 1	12	12	0	0%	0.667	168	171	172	176	178	178	171	3
Search	12	12	0	0%	0.667	169	171	172	177	179	179	172	3
Select	12	12	0	0%	0.667	168	171	173	176	178	178	171	3
Page 0	12	12	0	0%	0.667	168	171	172	175	177	177	171	2
Page 1	12	12	0	0%	0.667	167	171	171	175	177	178	170	3
Page 2	12	12	0	0%	0.667	167	171	171	175	178	179	171	3
Page 3	12	12	0	0%	0.667	167	172	172	175	177	178	171	3
Form	4	4	0	0%	0.222	168	169	170	170	170	170	169	1
Post	4	4	0	0%	0.222	168	171	172	172	172	172	171	2
Post Redirect 1	4	1	3	75%	0.222	168	171	171	172	172	172	170	1

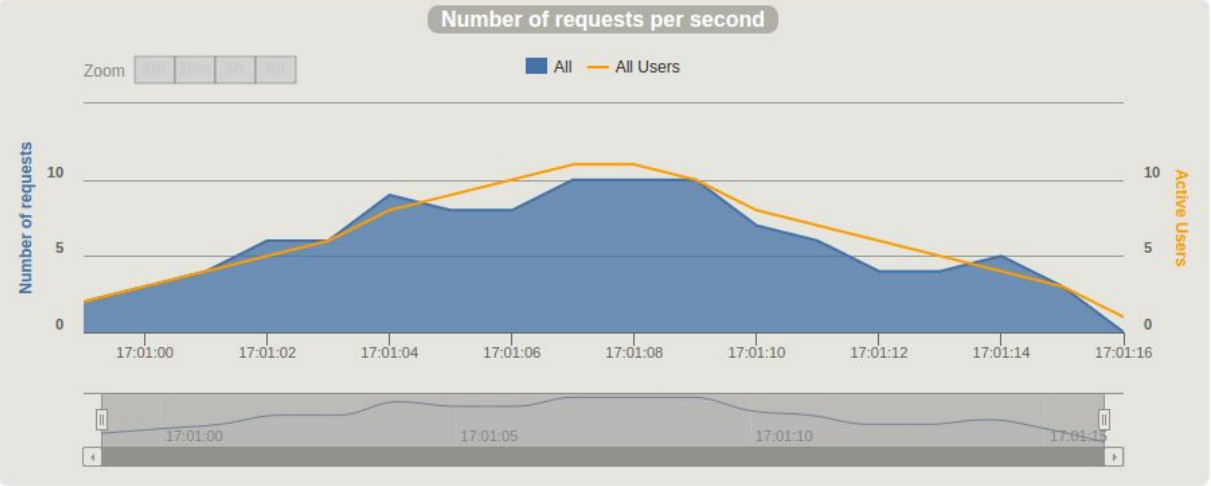
ERRORS

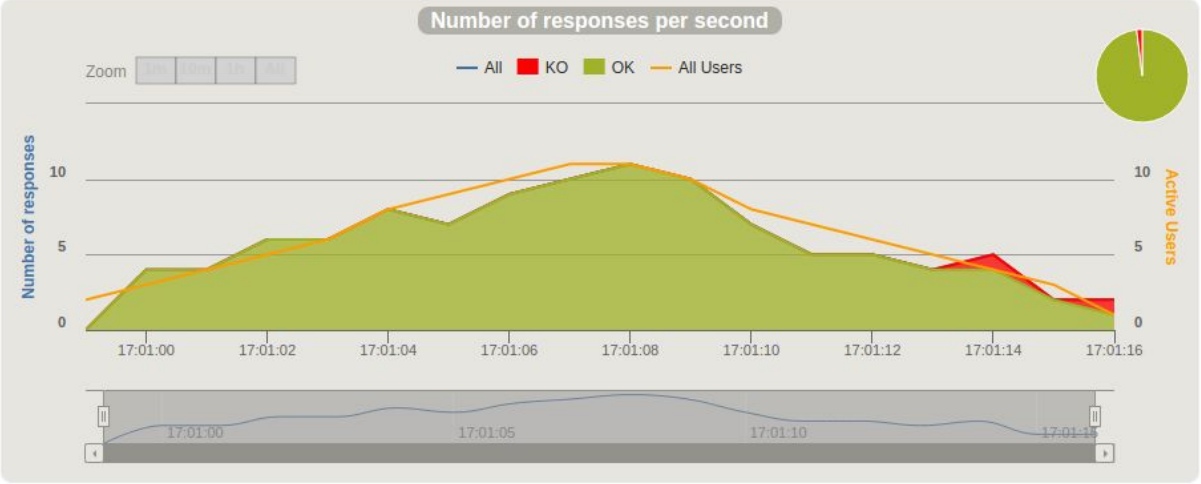
Error	Count	Percentage
status.find.is(201), but actually found 200	3	100 %











7. Quality Attributes

7.1 SCALABILITY

La arquitectura propuesta es escalable horizontalmente ya que gracias al balanceador de carga se pueden añadir más servidores que corran la aplicación, y el juego se seguirá comportando de una forma correcta, incluso un manejo de más peticiones.

7.2 AVAILABILITY

La arquitectura propuesta garantiza disponibilidad en cuanto a si un servidor en el cual está corriendo la aplicación deja de funcionar, los demás servidores pueden seguir atendiendo las solicitudes de los clientes, ya que el balanceador de carga redirecciona a los nuevos clientes a los servidores disponibles. Sin embargo esta arquitectura cuenta con dos puntos únicos de fallos, los cuales son los servidores Ubuntu ya que uno posee el servidor de mensajería Active Mq, y el otro posee el caché Redis y además el balanceador de carga Nginx. Una posible solución a esta problemática es contemplar estos servicios de manera individual por medio de distintos clusters, para garantizar una mejor disponibilidad del servicio sin puntos únicos de fallo.

7.3 SECURITY

En cuanto a seguridad, se propone manejar el acceso a direcciones url de la página de una mejor manera, ya que se puede acceder a la dirección /jugar.html y /select.html las cuales como no se está siguiendo el protocolo adecuado de registro de nombre anónimo pueden llegar a alterar el funcionamiento de la aplicación, por otra parte al no manejar un registro de usuarios, ya que estos son anónimos, la aplicación puede ser amenazada por ataques DOS o DDOS , haciendo que las salas se llenen de usuarios incorrectos y haciendo que el juego tenga jugadores “no reales” evitando que su jugabilidad sea la propuesta.