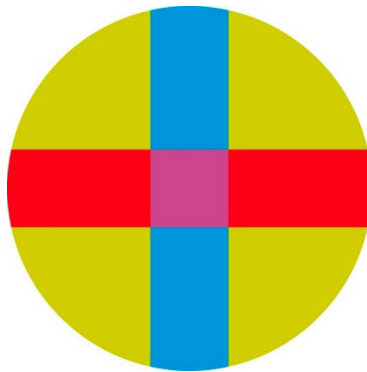


UNIVERSIDAD SAN PABLO - CEU

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA DE SISTEMAS DE INFORMACIÓN



TRABAJO FIN DE GRADO

**Diseño y Desarrollo de “Chischás!”:
Plataforma Web para Jugar Ajedrez en
Línea**

***Design and Development of “Chischás!”:
Web Platform for Playing Online Chess***

Autor: Miguel Gamboa Sánchez

Tutor: Álvaro Sánchez Picot

Junio 2025



UNIVERSIDAD SAN PABLO-CEU

ESCUELA POLITÉCNICA SUPERIOR

División de Ingeniería

Calificación del Trabajo Fin de Grado

Datos del alumno

NOMBRE: MIGUEL GAMBOA SÁNCHEZ

Datos del Trabajo

TÍTULO DEL PROYECTO: Diseño y Desarrollo de "Chischás!": Plataforma Web para Jugar Ajedrez en Línea

Tribunal calificador

PRESIDENTE:

FDO.:

SECRETARIO:

FDO.:

VOCAL:

FDO.:

Reunido este tribunal el ____/____/_____, acuerda otorgar al Trabajo Fin de Grado presentado por D./Dña. _____ la calificación de _____

Resumen

“Chischás!” es una aplicación web de partidas online de ajedrez diseñada para una experiencia de usuario rápida, simple, accesible y directa. El objetivo de la aplicación es permitir a sus usuarios echar partidas rápidas sin la necesidad de un registro obligatorio, simplemente enviando un enlace a su rival que el mismo genera. Aun así, cada usuario, si lo desea, puede crear un perfil para acceder a su historial de partidas, cambiarse el nombre, el correo electrónico o incluso la foto de perfil. Además, tiene fácil acceso a sus victorias, derrotas y tablas totales.

Node.js y Express son las bases de su desarrollo y están alojadas en una instancia de Amazon Web Services comunicada por HTTPS para garantizar su seguridad. Las decisiones de diseño basadas en su simplicidad dejan abiertas las puertas a un futuro redimensionado de la aplicación en base a su escalabilidad o aumento de funcionalidades.

Uno de los pilares del proyecto es su orientación hacia el código abierto. *Chischás!* dispone de un repositorio público en GitHub para que cualquier desarrollador que lo desee pueda estudiar o proponer nuevas funcionalidades, además de descargar del proyecto para alojarlo en su propio servidor.

Palabras Clave

Aplicación web, ajedrez, Node.js, Express, Amazon Web Services (AWS), HTTPS, GitHub, código abierto

Abstract

“Chischás!” is an online chess web application designed for a fast, simple, accessible, and straightforward user experience. The application's goal is to allow users to play quick games without mandatory registration, simply by sending a self-generated link to their opponent. However, each user can create a profile to access their game history, change their name, email address, or even their profile picture if they wish. Furthermore, they have easy access to their wins, losses, and draw totals.

Node.js and Express are the foundation of its development, and it is hosted on an Amazon Web Services instance connected via HTTPS to ensure security. Design decisions based on simplicity leave the door open for future resizing of the application based on scalability or increased functionality.

One of the pillars of the project is its open source orientation. Chischás! It has a public repository on GitHub so that any developer who wishes can study or propose new features, in addition to downloading the project to host it on their own server.

Keywords

Web application, chess, Node.js, Express, Amazon Web Services (AWS), HTTPS, GitHub, open source

Índice de contenidos

Capítulo 1 Introducción	10
1.1 Contexto del TFG	10
1.2 Objetivos.....	11
Capítulo 2 Gestión del proyecto	12
2.1 Modelo de ciclo de vida.....	12
2.2 Planificación.....	14
2.3 Presupuesto (si procede)	18
Capítulo 3 Análisis.....	21
3.1 Especificación de requisitos.....	21
3.2 Análisis de los Casos de Uso	24
3.3 Análisis de seguridad	25
3.4 Análisis desde la perspectiva del RGPD (si procede)	27
Capítulo 4 Diseño e implementación	29
4.1 Arquitectura del sistema	29
4.1.1 Arquitectura física	29
4.1.2 Arquitectura lógica	30
4.2 Diseño de datos	37
4.3 Interfaz de usuario.....	40
4.4 Diagrama de clases	44
4.5 Entorno de construcción.....	47
4.6 Referencia al repositorio de software	48
Capítulo 5 Validación del sistema.....	49
5.1 Plan de pruebas	49
Capítulo 6 Conclusiones y líneas futuras	52
6.1 Conclusiones	52

6.2 Líneas futuras.....	53
Capítulo 7 Bibliografía	54
Anexo I - Configuración de NGINX y HTTPS:.....	57
Anexo II – Librerías utilizadas	59
Glosario de términos	62

Índice de figuras

<i>Ilustración 1 Fases de un ciclo de vida incremental.....</i>	<i>12</i>
<i>Ilustración 2 Cronograma de Gantt PREDICCIÓN</i>	<i>16</i>
<i>Ilustración 3 Cronograma de Gantt FINAL/REAL</i>	<i>17</i>
<i>Ilustración 4 Diagrama de casos de usos.....</i>	<i>24</i>
<i>Ilustración 5 Región servidor AWS.....</i>	<i>28</i>
<i>Ilustración 6 Diagrama de arquitectura física</i>	<i>29</i>
<i>Ilustración 7 Diagrama de capas lógicas</i>	<i>31</i>
<i>Ilustración 8 Ejemplo ilustrativo de Reverse Proxy</i>	<i>32</i>
<i>Ilustración 9 Captura de pantalla de interfaz de seguridad AWS.....</i>	<i>34</i>
<i>Ilustración 10 Dirección IP elástica</i>	<i>34</i>
<i>Ilustración 11 Direcciones DNS Namecheap.....</i>	<i>35</i>
<i>Ilustración 12 Candado Hhttps en navegador</i>	<i>35</i>
<i>Ilustración 13 Certificado digital de www.chischas.xyz.....</i>	<i>36</i>
<i>Ilustración 14 Diagrama de diseño de datos</i>	<i>37</i>
<i>Ilustración 15 Diagrama del diseño de la interfaz de usuario</i>	<i>40</i>
<i>Ilustración 16 Home de Chischás!.....</i>	<i>41</i>
<i>Ilustración 17 Menú de juego de Chischás!</i>	<i>41</i>
<i>Ilustración 18 Partida en línea de Chischás!</i>	<i>42</i>
<i>Ilustración 19 Formulario de registro de Chischás!</i>	<i>42</i>
<i>Ilustración 20 Formulario de Login de Chischás!</i>	<i>43</i>
<i>Ilustración 21 Perfil de usuario de Chischás! Ejemplo Elisabeth (perfil invitado)</i>	<i>44</i>
<i>Ilustración 22 Diagrama de clases de la aplicación.....</i>	<i>44</i>
<i>Ilustración 23 Resumen de prueba de carga</i>	<i>50</i>
<i>Ilustración 24 Informe de rendimiento HTTP: tiempos de respuesta y desglose por endpoint</i>	<i>50</i>

Índice de tablas

<i>Tabla 1 Predicción de planificación</i>	<i>15</i>
<i>Tabla 2 Planificación final/real.....</i>	<i>17</i>
<i>Tabla 3 Costes estimados futuros Recursos Humanos</i>	<i>18</i>
<i>Tabla 4 Costes estimados infraestructura anual actual</i>	<i>19</i>
<i>Tabla 5 Costes estimados licencias de Software</i>	<i>19</i>
<i>Tabla 6 Coste total estimado.....</i>	<i>20</i>
<i>Tabla 7 Requisitos funcionales</i>	<i>23</i>
<i>Tabla 8 Requisitos no funcionales</i>	<i>23</i>
<i>Tabla 9 Resultados de prueba de carga</i>	<i>51</i>
<i>Tabla 10 Estimación máximos usuarios en partida</i>	<i>51</i>

Capítulo 1

Introducción

1.1 Contexto del TFG

Anteriores experiencias personales con aplicaciones de ajedrez en línea (chess.com, ajedrezonline.com...) las cuales eran tediosas y no permitían jugar con amigos mediante una invitación rápida, sin tener que pasar algún registro o instalar algún software, hacen que surja la necesidad personal de realizar “Chischás!”.

Según la ONU, un 70% de la población adulta, en algún momento de su vida, ha jugado una partida de ajedrez, esto equivale a 605 millones de personas (Naciones Unidas, 2025). Ocasiones como la pandemia (Covid-19) nos demostraron que este deporte es algo relevante en nuestra sociedad y plataformas como Chess.com fueron una gran aportación para sustentar la paciencia de muchas personas en tal crisis, incrementando el juego en línea hasta en un 40% (El País, 2025).

Por eso y por más cosas nace “Chischás!” como Trabajo de Fin de Grado situándose en el ámbito del desarrollo web, buscando posibilitar la conexión entre dos jugadores de manera simple y rápida, permitiendo, si el jugador lo desea, saltarse el registro.

En un entorno digital cada vez más saturado de plataformas complejas o sobrecargadas de funcionalidades, esta propuesta busca recuperar la inmediatez y la simplicidad, centrándose en lo esencial del juego: el tablero, las piezas y los jugadores.

En la parte técnica, es su despliegue en la nube mediante AWS, su backend apoyado en Node.js y Express, y la integración lógica de las reglas del ajedrez mediante la asistencia de algunas librerías específicas, lo que le aportan el peso a su aplicación práctica. Además, el ser código abierto, permite el despliegue, análisis y edición por parte de la comunidad, fomentando así la solicitud de futuras aportaciones, un

despliegue descentralizado por parte de la comunidad y el uso del código para aprendizaje.

El enlace al repositorio está situado en la sección 4.6.

1.2 Objetivos

El objetivo principal, desde el primer instante, ha sido desarrollar una aplicación web funcional, accesible y rápida para jugar partidas de ajedrez en línea entre dos personas, con el funcionamiento que se ha comentado en el resumen.

Para alcanzar este objetivo general, se han definido los siguientes objetivos específicos:

1. Diseñar una interfaz clara y funcional, adaptada exclusivamente para su uso en ordenadores, priorizando la simplicidad visual y la facilidad de uso.
2. Implementar la lógica del ajedrez en tiempo real, utilizando tecnologías como Node.js para el servidor y WebSockets para la comunicación bidireccional entre los jugadores liderada por el servidor.
3. Incorporar un sistema de usuarios opcional, que permita a quienes lo deseen crear un perfil con nombre, correo electrónico y foto, así como acceder a su historial de partidas (victorias, derrotas y tablas).
4. Desplegar la aplicación en la nube mediante Amazon Web Services (AWS), con el fin de garantizar disponibilidad, escalabilidad y buen rendimiento en todo momento.
5. Habilitar un dominio web propio, accesible a través de la dirección www.chischas.xyz, para facilitar el acceso a la plataforma desde cualquier navegador de escritorio.

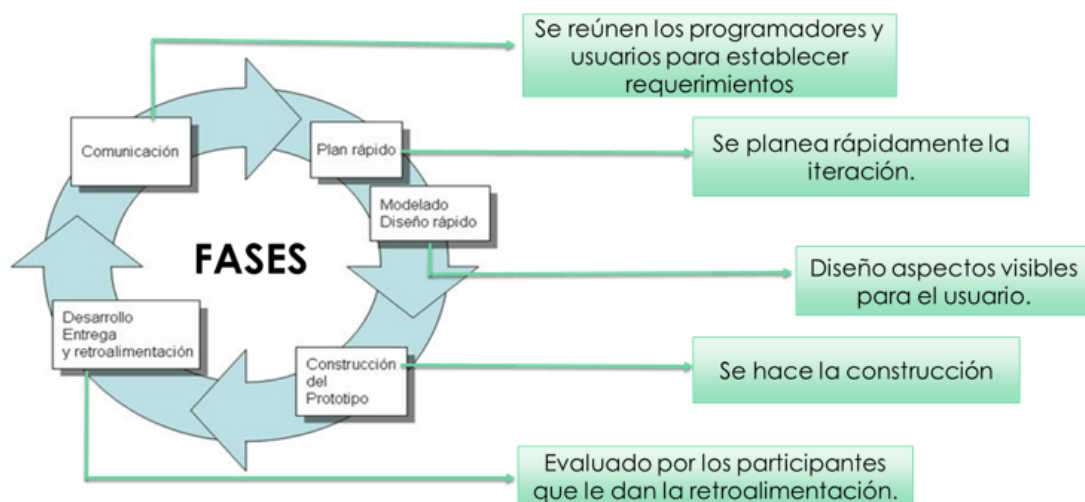
Capítulo 2

Gestión del proyecto

2.1 Modelo de ciclo de vida

El modelo de ciclo de vida se basa en una rueda incremental como muestra la ilustración 1, permitiendo construir la aplicación desde el primer momento y haciendo el proceso progresivo para permitir añadir funcionalidades en ciclos sucesivos, apoyados siempre en una base en funcionamiento mínima.

Ilustración 1 Fases de un ciclo de vida incremental



Nota. es importante respetar cada fase del ciclo de vida, aunque trabajando individualmente pueda variar ligeramente. Tomado de (Comparasoftware, 2025)

En mi caso, al ser un único programador en el proyecto, la primera etapa de comunicación y requerimientos no existía u ocurría puntualmente con mi tutor de TFG. Ocurría lo mismo con la evaluación con retroalimentación.

En el cronograma final del apartado 2.2 podemos ver como el proyecto se ha desarrollado en iteraciones diferenciadas, permitiendo implementar primero lo esencial y, finalmente, incorporar nuevas capacidades y mejoras continuas.

Las razones que han determinado el uso de este modelo de ciclo de vida incremental han sido bastantes:

Entrega temprana de versiones funcionales: Desde las primeras etapas se construyó una versión básica que permitía jugar partidas en línea entre dos invitados, sin necesidad de registro, garantizando así una validación temprana del núcleo de la aplicación.

Desarrollo de funcionalidades de manera progresiva: tras finalizar la base inicial se integró escalonadamente la lógica del ajedrez completa junto a otros elementos clave (sistema de perfiles, inicio de sesión, registro, guardado de partidas, conexión a la BDD...)

Fase de integración y despliegue: Tras finalizar las funciones básicas, se llevó a cabo la conexión con el dominio propio (www.chischas.xyz) y el despliegue en la nube a través de AWS. En esta etapa también se elaboró la documentación del proyecto, asegurando que cada avance se integrara de forma ordenada y operativa.

Correcciones y mejoras: En las últimas fases del desarrollo, se dedicaron esfuerzos a corregir errores detectados, mejorar el rendimiento de la plataforma y añadir funciones específicas, como la gestión de sesiones en el servidor y la inclusión de relojes para las partidas.

Pensamiento largoplacista: Gracias a la metodología incremental, el proyecto queda preparado para crecer. Esto permitirá añadir nuevas modalidades de juego o implementar características sociales adicionales sin afectar al buen funcionamiento de la aplicación.

La planificación general se realizó durante las primeras fases, pero, la mayor parte del desarrollo consistió en ciclos de implementación, prueba manual y validación SIN TDD siguiendo las características del modelo incremental.

2.2 Planificación

La planificación de este proyecto se apoya en la Métrica 3 (Digital, s.f.) debido al uso de herramientas como el diagrama de Gantt realizando uno al principio de todo el proyecto (ilustración 2) y otro que iba reflejando la realidad a medida que iba avanzando (ilustración 3) .

Fue importante identificar las tareas de máxima prioridad para el desarrollo de “Chischás!” para así desglosarlas en subtareas que abarcan todo el proyecto.

Las tareas más importantes fueron:

1. Estructura básica Node.js y flujos cliente-servidor (10-17 febrero 2025).
2. Diseño funcional web y presentación CSS (12 febrero - 6 mayo 2025).
3. Integración de la lógica de ajedrez y gestión de usuarios (marzo-abril 2025).
4. Implementación de guardado de partidas y gestión de sesiones (abril-mayo 2025).
5. Despliegue en la nube (AWS) e integración del dominio (mayo 2025).
6. Corrección de errores y documentación final (mayo-junio 2025).

La planificación se vio obligada a intercalar ciertas dependencias entre tareas, como por ejemplo:

7. La Implementación de invitados mediante link (19 febrero - 4 marzo 2025) dependió de la correcta configuración de los flujos cliente-servidor.
8. La Lógica de ajedrez (30 marzo - 13 abril 2025) debía estar completada antes de implementar el guardado de partidas.
9. El despliegue del dominio y sesiones (abril-mayo 2025) se abordó una vez estabilizadas las funcionalidades principales.

La estimación de esfuerzo y recursos se basó en la complejidad técnica de cada tarea teniendo en cuenta experiencias previas. Entre marzo y abril fueron los meses de mayor esfuerzo de implementación. Mayo y junio fueron meses de integración, corrección y documentación. A continuación, los diagramas de Gantt con sus respectivas tablas.

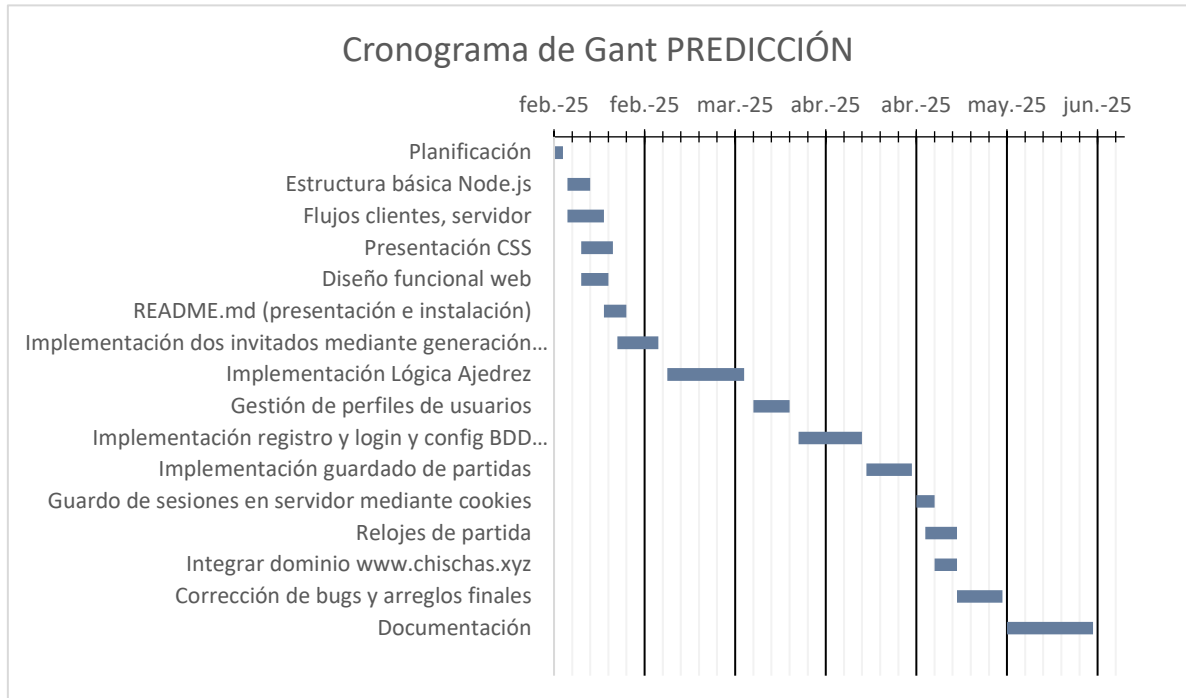
PREDICCIÓN DE LA PLANIFICIACIÓN

La tabla 1 y la ilustración 2 muestra la predicción de planificación, realizada el día 6 de febrero tras dar comienzo a este proyecto. La intención era realizar cada tarea en su fecha para así poder realizar la entrega de manera eficiente. Durante el tiempo, las fechas se fueron distorsionando un poco.

Tabla 1 Predicción de planificación

Tarea	Inicio	Número días	Fin
Planificación	06/02/2025	2	08/02/2025
Estructura básica Node.js	09/02/2025	5	14/02/2025
Flujos clientes, servidor	09/02/2025	8	17/02/2025
Presentación CSS	12/02/2025	7	19/02/2025
Diseño funcional web	12/02/2025	6	18/02/2025
README.md (presentación e instalación)	17/02/2025	5	22/02/2025
Implementación dos invitados mediante generación de link	20/02/2025	9	01/03/2025
Implementación Lógica Ajedrez	03/03/2025	17	20/03/2025
Gestión de perfiles de usuarios	22/03/2025	8	30/03/2025
Implementación registro y login y config BDD MongoDB	01/04/2025	14	15/04/2025
Implementación del guardado de partidas	16/04/2025	10	26/04/2025
Guardo de sesiones en servidor mediante cookies	27/04/2025	4	01/05/2025
Relojes de partida	29/04/2025	7	06/05/2025
Integrar dominio www.chischas.xyz	01/05/2025	5	06/05/2025
Corrección de bugs y arreglos finales	06/05/2025	10	16/05/2025
Documentación	17/05/2025	19	05/06/2025

Ilustración 2 Cronograma de Gantt PREDICCIÓN



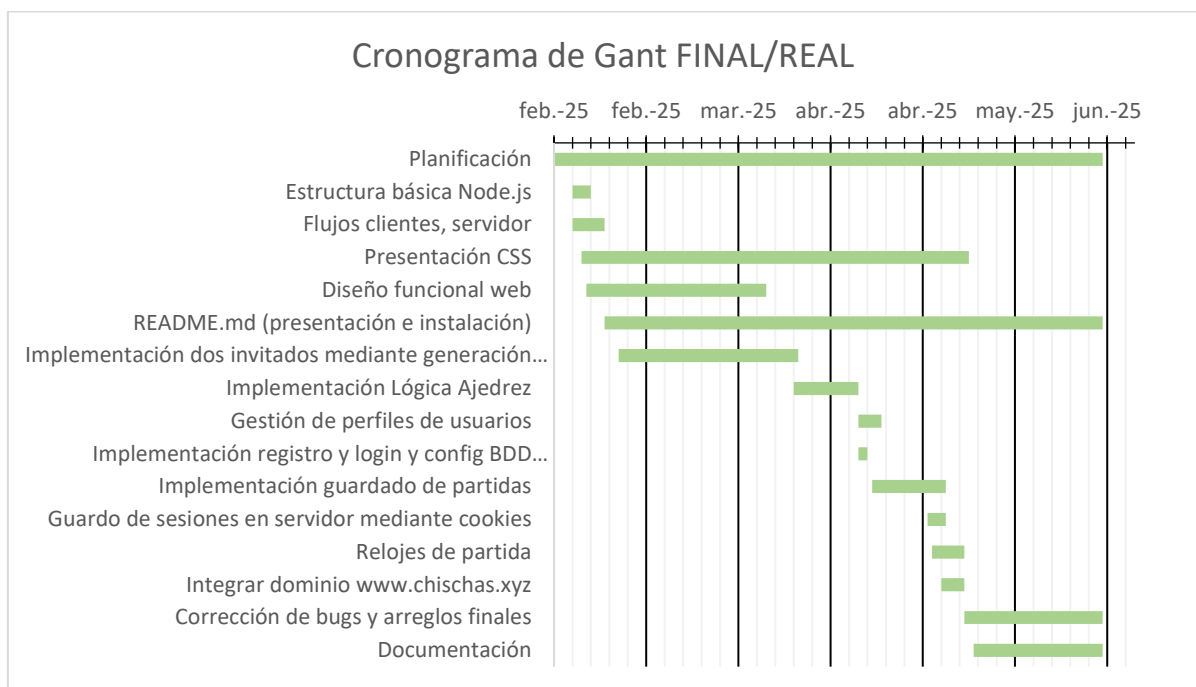
PLANIFICACIÓN FINAL/REAL

La Tabla 2 y la ilustración 3 muestran las predicciones pero con la distorsión temporal aplicada. Como se ve el inicio es el mismo, pero a medida que avanza, el calendario se va readaptando. Algunas de las tareas tienen una fecha desorbitada pero en parte es porque esto no ha sido un proyecto a tiempo completo y ha tenido sus parones, con temporadas de avances muy pequeños.

Tabla 2 Planificación final/real

Tarea	Inicio	Número días	Fin
Planificación	06/02/2025	119	05/06/2025
Estructura básica Node.js	10/02/2025	4	14/02/2025
Flujos clientes, servidor	10/02/2025	7	17/02/2025
Presentación CSS	12/02/2025	84	07/05/2025
Diseño funcional web	13/02/2025	39	24/03/2025
README.md (presentación e instalación)	17/02/2025	53	11/04/2025
Implementación dos invitados mediante generación de link	20/02/2025	39	31/03/2025
Implementación Lógica Ajedrez	30/03/2025	14	13/04/2025
Gestión de perfiles de usuarios	13/04/2025	5	18/04/2025
Implementación registro y login y config BDD MongoDB	13/04/2025	2	15/04/2025
Implementación del guardado de partidas	16/04/2025	16	02/05/2025
Guardo de sesiones en servidor mediante cookies	28/04/2025	4	02/05/2025
Relojes de partida	29/04/2025	7	06/05/2025
Integrar dominio www.chischas.xyz	01/05/2025	5	06/05/2025
Corrección de bugs y arreglos finales	06/05/2025	30	05/06/2025
Documentación	08/05/2025	28	05/06/2025

Ilustración 3 Cronograma de Gantt FINAL/REAL



2.3 Presupuesto (si procede)

El desarrollo de “Chischás!” solo ha tenido en cuenta los costes del tiempo dedicado en su avance y el precio del dominio. Sin embargo, conviene apuntar a largo plazo, así que calcularemos los costes totales de la aplicación, tanto de su mantenimiento como de su desarrollo de nuevas funcionalidades para cubrir una escalabilidad total. Se estimarán: costes de recursos humanos, infraestructura y licencias de software. Finalmente, el total.

1. Recursos humanos:

La tabla 3 muestra cual sería el coste de los recursos humanos en caso de querer desarrollar “Chischás!” desde cero y llevarla a un nivel competitivo en un posible futuro.

Tabla 3 Costes estimados futuros Recursos Humanos

Perfil	Tareas	Dedicación estimada	Coste hora	Total
Desarrollador Full Stack (Node.js, WebSocket's, MongoDB)	Desarrollo backend, frontend y despliegue	300 h	25 €/h	7.500 €
Diseñador UI/UX	Diseño de interfaz y experiencia de usuario	50 h	20 €/h	1.000 €
Administrador de Sistemas (DevOps)	Configuración servidores, dominio, SSL, despliegue	30 h	30 €/h	900 €
Total Recursos Humanos				9.400 €

2. Infraestructura

La tabla 4 expone cuales son los actuales costes de infraestructura de la aplicación. Esto representa bastante bien la filosofía descentralizada de “Chischás!” ya que busca ser un sistema que cualquiera pueda desplegar al ser código abierto y no tener costes elevados, además de ser fácilmente sustituibles.

Tabla 4 Costes estimados infraestructura anual actual

Concepto	Descripción	Periodicidad	Precio	Total (anual)
Dominio web (.xyz)	Registro de dominio	Anual	1,78 €	1,78 €
Instancia AWS EC2 (el plan de “Chischás!” es gratuito debido que lo proporciona la universidad con AWS Academy)	Servidor básico para producción (t2.micro)	Anual	10 €	120 €
Certificado SSL (Let’s Encrypt)	Gratuito	Anual	0 €	0 €
Almacenamiento MongoDB Atlas	Plan gratuito inicialmente	Mensual	0 €	0 €
Total Infraestructura Anual				121,78 €

Es importante tener en cuenta que los costes de la infraestructura irán escalando a medida que aumente la ambición de nuestros objetivos, como puede ser aumentar la cantidad de usuarios o el tráfico, pasando a una MongoDB Atlas de pago o a otros tipos de EC2.

3. Licencias de Software

El proyecto está planteado desde un inicio con herramientas de código abierto, así que no existe ningún cargo financiero en licencias de software. En la tabla 5 se puede ver que las únicas licencias son Visual Studio Code y GitHub.

Tabla 5 Costes estimados licencias de Software

Concepto	Descripción	Coste	Total
Visual Studio Code	Entorno de desarrollo (gratuito)	0 €	0 €
GitHub	Repositorio público (gratuito)	0 €	0 €
Total Licencias			0 €

4. Coste Total Estimado

Teniendo en cuenta todo lo anterior, la tabla 6 muestra como aquel que quisiese hacerse cargo de “Chischás!” y darle un valor estimado inicial podría establecer la cifra inicial basándose en el total del primer año.

Tabla 6 Coste total estimado

Concepto	Coste
Desarrollo inicial	9.400,00 €
Infraestructura anual	121,78 €
Licencias	0,00 €
Total primer año	9.521,78 €

Capítulo 3

Análisis

3.1 Especificación de requisitos

Los requisitos del sistema han sido una guía importante a la hora de encaminar el proyecto. Se establecieron desde el inicio visualizando una aplicación funcional que satisfacía la idea inicial. Al principio, “Chischás!” iba a ser una plataforma más compleja, pero, tras la planificación, la realidad salió a la luz y se decidió poner en requisitos lo más indispensable, sin florituras.

Funcionalidades del sistema

Los usuarios pueden jugar partidas de ajedrez en línea sin necesidad de registro previo. Compartir un enlace generado automáticamente para invitar a un rival. Además, crear un perfil personal para acceder al historial de partidas, modificar datos personales (nombre, correo, foto de perfil) y consultar estadísticas (victorias, derrotas, tablas).

Administración del sistema:

Registro de partidas para análisis posterior. Se deja abierta la opción de implementar en el futuro un sistema de moderación y control de comportamiento.

Interfaz de usuario

El sistema deberá presentar una interfaz gráfica que permita al usuario acceder a las funcionalidades principales (crear partida, unirse a partida) en un máximo de 3 clics desde la página principal, sin necesidad de registro previo.

La aplicación contará con un nivel básico de accesibilidad que permite la interacción con todo el contenido presente. No se garantiza el cumplimiento de algún estándar formal común.

Seguridad

Uso de HTTPS para cifrado de comunicaciones. Autenticación mínima para usuarios registrados (correo + contraseña). Actualmente, no se aplican controles del Esquema Nacional de Seguridad ni de normas ISO, pero se planea una evaluación de riesgos. El apartado 3.3 hace un análisis de seguridad.

Protección de datos

- a. En desarrollo: cumplimiento básico del RGPD y LOPDGDD. Apartado 3.4.
- b. Recogida mínima de datos personales (nombre, correo electrónico).
- c. Pendiente la implementación de funciones como exportación y eliminación de datos.

Requisitos sobre entorno tecnológico y de comunicaciones

- a. Backend: Node.js + Express.
- b. Frontend: HTML, CSS, JavaScript, EJS.
- c. Infraestructura: Amazon Web Services (EC2, S3).
- d. Base de datos: MongoDB (opcional para historial de usuarios registrados).
- e. Comunicación segura mediante HTTPS.
- f. Protocolo WebSocket para partidas en tiempo real.

Por lo cual, reduciendo al mínimo los planes de esta aplicación, llegamos a los siguientes requisitos funcionales (tabla 7). Estos serán satisfechos a medida que la memoria avanza, ya que, no es simplemente código JavaScript, sino que para hacerlos posibles, esto requiere una infraestructura más elaborada de más de un nivel.

Tabla 7 Requisitos funcionales

Ref.	Descripción
RF01	Los usuarios deben poder solicitar un enlace de partida sin registro
RF02	El servidor debe poder generar el enlace compatible de partida
RF03	El usuario debe de tener la posibilidad de un registro opcional con correo electrónico, usuario y contraseña
RF04	El usuario debe poder visualizar su historial de partidas
RF05	El usuario debe poder visualizar el número de partidas ganadas, perdidas y empatadas.
RF06	El usuario debe poder modificar los datos de su perfil
RF07	Los usuarios deben poder jugar partidas con normalidad tanto si están registrados como si no
RF08	Los usuarios deben poder chatear con su rival durante la partida

Además, también se tuvieron en cuenta los requisitos no funcionales. La tabla 8 muestra cualidades que el sistema debe cumplir pero fuera de su comportamiento.

Tabla 8 Requisitos no funcionales

Categoría	Requisito
Rendimiento	Tiempos de respuesta inferiores a 500 ms para la mayoría de las acciones de usuario (inicio de partida, movimientos).
Capacidad	Escalable para manejar hasta 6000 partidas simultáneas en la versión actual.
	Capacidad futura estimada: 20000 partidas concurrentes tras optimización y escalado horizontal.
Seguridad	Comunicación segura mediante HTTPS.

En el apartado de validación (5.1), se ha realizado una prueba “Ramp Up” que muestra cómo estos requisitos se cumplen finalmente. Excepto la de seguridad, que también se cumple, pero su implementación se explica en múltiples apartados.

3.2 Análisis de los Casos de Uso

La siguiente ilustración revela los casos principales de la plataforma web. Además, se identifican las funcionalidades más importantes del sistema desde las perspectivas de los usuarios diseñados.

Ilustración 4 Diagrama de casos de usos



Nota. El diagrama refleja las interacciones entre los distintos tipos de usuarios y las funcionalidades disponibles. Elaboración propia.

Actores del caso de uso:

Usuario registrado: este usuario dispone de una cuenta creada por el mismo con anterioridad y tiene acceso exclusivo a la edición y visualización de su perfil, el cierre de sesión, y las funcionalidades comunes (unirse a partida y crear partida)

Usuario invitado: este usuario accede al sistema web sin previamente haber creado una cuenta, teniendo acceso exclusivo a registrarse e iniciar sesión, además de las funciones comunes ya citadas.

Casos de uso (se usará RF como relación a los requisitos funcionales del apartado 3.1):

- Iniciar sesión permite al usuario autenticarse y cumple el requisito funcional RF03.
- Registrarse permite crear una cuenta de usuario de manera voluntaria, satisfaciendo el requisito RF03.
- Crear partida permite generar un enlace compartible, validando los requisitos RF01, RF02 y RF07.
- Unirse a la partida permite acceder a un estado de juego mediante el enlace, aportando a los requisitos RF01, RF02 y RF07.
- Editar perfil permite modificar los datos del usuario, aportando y cumpliendo el requisito RF06.
- Ver perfil permite ver el recuento de partidas, resultados e historial, cumpliendo los requisitos RF04 y RF05.
- Cerrar sesión permite salir de la cuenta y eliminar las cookies. Es una función implícita dentro del flujo de usuarios.

3.3 Análisis de seguridad

La implementación y configuración de las medidas de seguridad, no las trataremos en este apartado, esa sección estará ubicada en el apartado 4.1.2 arquitectura lógica.

El análisis de riesgo realizado consta con cinco divisiones fundamentales: autenticidad, confidencialidad, integridad, disponibilidad y trazabilidad. Gracias a esta alineación podremos cubrir e identificar el máximo número de posibles amenazas para proponer una conducta de mitigación acertada.

Autenticidad:

Riesgo: suplantación de identidad de registrados.

Medidas: plan de autenticación multifactorial mediante autenticación de campos implementados (correo y contraseña), además de futura validación del correo electrónico al registrarse.

Confidencialidad:

Riesgo: acceso desautorizado a datos confidenciales de usuarios

Medidas: Cifrado de datos en tránsito con HTTPS. Acceso a los datos personales restringido al usuario correspondiente. Contraseñas cifradas en la BDD mediante Bcrypt

Integridad:

Riesgo: Alteración desautorizada de partidas o perfiles

Medidas: Validación del servidor en los estados de juego. Control de inyecciones mediante el guardado posterior al juego de partidas con id propia y fecha, además, limitación de conexión a la BDD

Disponibilidad:

Riesgo: Ataques DoS (Denegación de servicio) o sobrecarga de recursos.

Medidas: Posibilidad de balanceo en AWS

Trazabilidad

Riesgo: No se podría saber quién ha hecho qué si pasa algo raro.

Medidas: De momento, se guardan datos básicos como cuándo alguien inicia sesión o juega una partida. Más adelante, se quiere añadir un registro más completo y seguro para evitar que se puedan tocar esos datos.

3.4 Análisis desde la perspectiva del RGPD (si procede)

Este apartado no procede y además conviene explicar por qué. Como indica el glosario de términos el RGPD es el Reglamento General de Protección de Datos. Este reglamento es exclusivo para la protección de usuarios europeos. El servidor de “Chischás!” está en Virginia del Norte, América. Esto hace que en principio el reglamento no se aplique a “Chischás!”. Aun así, este sí protege a todo usuario europeo que almacene sus datos a pesar de lo dicho anteriormente.

Es por esto por lo que este apartado genera un margen de duda, pero, se concluye aclarando que “Chischás!” solo almacena el correo electrónico como dato personal, el nombre y una contraseña cifrada. Estos datos no son procesados con fines comerciales o de perfilado. Aun así, se han tomado medidas para satisfacer el principio de responsabilidad proactiva.

A continuación, una cita de la AEPD (Agencia Española de Protección de Datos) para dejar claro que es el principio de responsabilidad proactiva:

¿Qué es el principio de responsabilidad proactiva?

El RGPD describe este principio como la necesidad de que el responsable del tratamiento aplique medidas técnicas y organizativas apropiadas a fin de garantizar y poder demostrar que el tratamiento es conforme con el Reglamento.

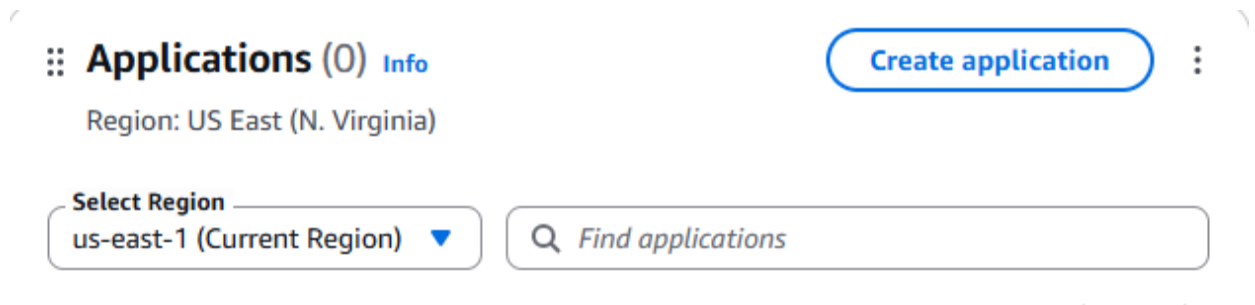
En términos prácticos, este principio requiere que las organizaciones analicen qué datos tratan, con qué finalidades lo hacen y qué tipo de operaciones de tratamiento llevan a cabo.

A partir de este conocimiento deben determinar de forma explícita la forma en que aplicarán las medidas que el RGPD prevé, asegurándose de que esas medidas son las adecuadas para cumplir con el mismo y de que pueden demostrarlo ante los interesados y ante las autoridades de supervisión.

En síntesis, este principio exige una actitud consciente, diligente y proactiva por parte de las organizaciones frente a todos los tratamientos de datos personales que lleven a cabo.

(Agencia Española de Protección de Datos, 2025)

Ilustración 5 Región servidor AWS



Tomado de (propia, Vista de la región del servidor en AWS, 2024)

Capítulo 4

Diseño e implementación

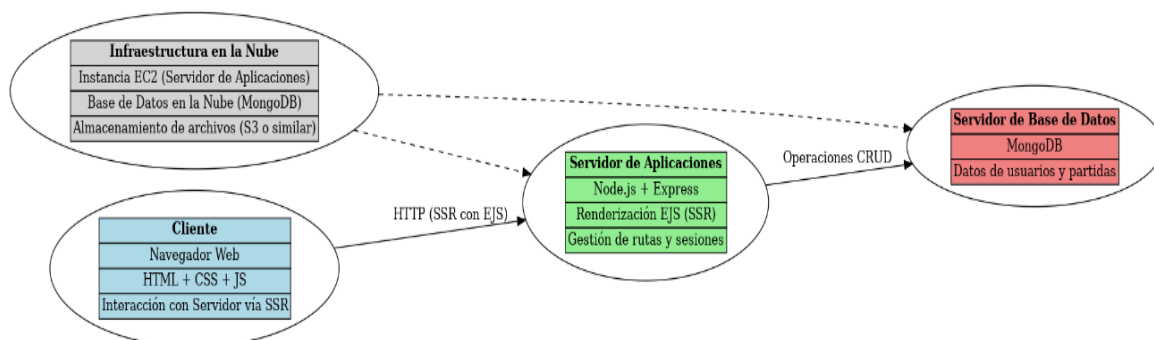
4.1 Arquitectura del sistema

Este apartado expone la arquitectura general “Chischás!”, abarcando dos diferentes perspectivas (arquitectura física y arquitectura lógica) que permiten el funcionamiento completo del sistema. La arquitectura destaca por simpleza en ambos ámbitos.

4.1.1 Arquitectura física

La arquitectura física viene reflejada en la Ilustración 6, que sigue un modelo clásico cliente-servidor apoyado en tecnologías escalables en la nube. Contiene 4 elementos principales: cliente, servidor de aplicaciones, servidor de base de datos e infraestructura en la nube.

Ilustración 6 Diagrama de arquitectura física



Nota. Elaboración propia.

A continuación, se explicarán los componentes físicos del sistema.

Una instancia Amazon EC2 t2.micro protagoniza el despliegue de la aplicación. La instancia contiene el servidor de la aplicación y la MongoDB donde se almacenan los

jugadores y partidas, de esto hablaremos más a fondo en el apartado 4.2. Su funcionamiento implica múltiples componentes físicos y lógicos haciendo su trabajo de manera simultánea.

La instancia EC2 t2.micro pertenece al servicio gratuito de Amazon Web Service y proporciona un servidor con un vCPU (es un procesador virtual compartido), 1 GB de memoria RAM (suficiente para partidas de ajedrez basadas en WebSockets) y 10 GB de almacenamiento persistente EBS (Elastic Block Store).

Namecheap proporciona el servidor DNS externo para la resolución de nombre de dominio. Se entrará más en detalle en el siguiente apartado. Además, el Security Group el cual veremos en el apartado 4.1.2, actúa como un firewall que filtra el tráfico entrante y saliente hacia la instancia.

4.1.2 Arquitectura lógica

La instancia está desplegada desde una Virtual Private Cloud la cual está dentro de una subred pública. Es importante resaltar que, para que nuestro DNS apunte bien al servidor, ha sido necesario configurar un IP elástica. Esta dirección está vinculada de manera permanente a la instancia, permitiendo que después de un reinicio, una caída o un cambio de red, la IP se mantenga exactamente igual evitando que los clientes obtener un “ERR_CONNECTION_REFUSED” desde su navegador (si la IP responde, pero no hay nada escuchando en el puerto 80/443).

La parte lógica se puede dividir en tres capas: presentación, lógica de negocio y acceso de datos. Esta elección permite un control total sobre el entorno de ejecución, una fácil escalabilidad y una gestión directa de la seguridad.

La **capa de presentación** es la que se encarga de gestionar la interacción del usuario mediante la interfaz visible gracias a los HTML, CSS y JavaScript. Utiliza un sistema de renderizado SSR (server-side) para que los EJS (Embedded JavaScript Templates) puedan ejercer su función, haciendo que los HTML de cada usuario muestren lo conveniente de manera personalizada. Los formularios de registro, inicio de sesión, solicitud de creación de link de partida... son gestionados por esta capa mediante métodos GET/POST. Las partidas en tiempo real funcionan a través de WebSockets, haciendo instantánea la actualización de movimientos sin necesidad de actualizar la página.

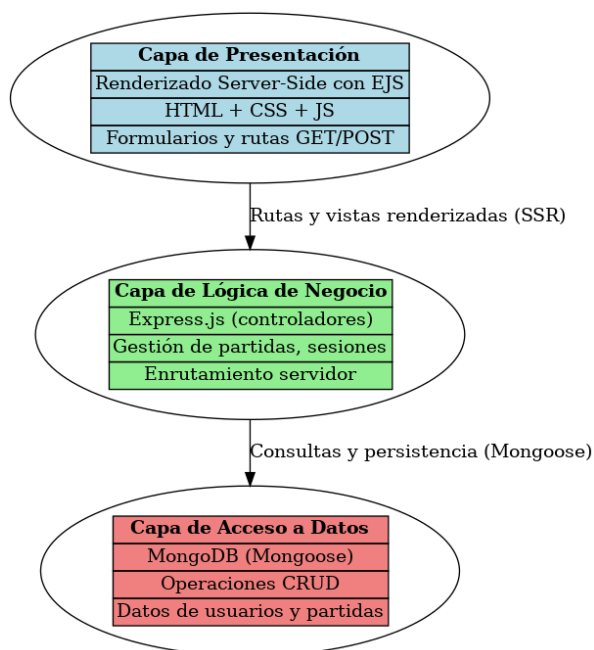


Ilustración 7 Diagrama de capas lógicas

La **capa de lógica de negocio** se sustenta gracias a Node.js con el entorno de desarrollo Express.js, para así gestionar la creación de partidas, sesiones de usuarios registrados, enrutamiento, validación de movimientos, etc. Los sockets también tienen importancia en esta capa debido que no solo el cliente los usa, sino que el servidor también para poder llegar al cliente rival.

La capa de acceso a datos gestiona la persistencia de los datos mediante MongoDB, con operaciones CRUD implementadas en el código gracias a la librería Mongoose. Esto garantiza la integridad y seguridad de los datos almacenados en la EC2 sin exponer el puerto al exterior

4.1.2.1 Componentes lógicos y software en la instancia

A continuación, describiremos los componentes lógicos y software que alberga este sistema web con su instancia EC2, así como su configuración básica y servicio de red. Parte está en el anexo I, este será citado más adelante.

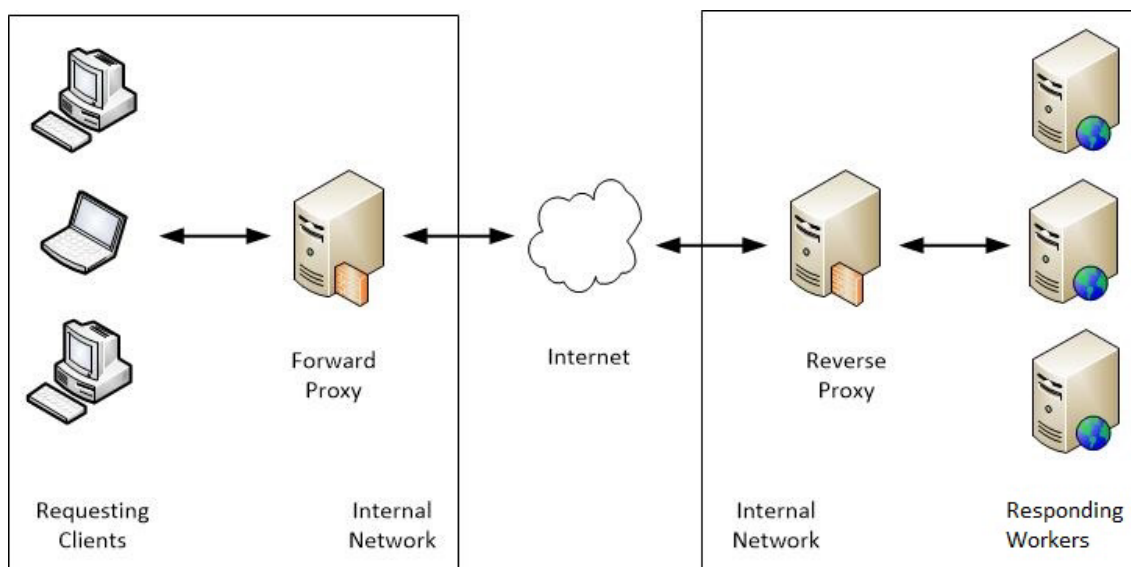
Este consta de una dirección IP pública asignada 34.199.73.164, con resolución DNS pública. Asociada a una VPC y a una Elastic IP para garantizar disponibilidad constante.

Dentro de la instancia EC2 encontramos desplegados los siguientes servicios instalados en una Ubuntu Server:

Aplicación de Node.js (servidor): apoyado en Express y Socket.IO, se encarga de gestionar las rutas web, la comunicación en tiempo real de partidas mediante WebSockets y la lógica de la aplicación.

Un servidor inverso NGINX para gestionar el tráfico HTTP/HTTPS y redirigirlo a la aplicación de Node.js mediante un proxy inverso. Se pueden encontrar más detalles sobre la configuración en el anexo I.

Ilustración 8 Ejemplo ilustrativo de Reverse Proxy



Nota. se puede ver cómo el reverse proxy también puede actuar de balanceador en caso de tener múltiples instancias. Tomado de (Microsoft, 2025)

Let's Encrypt proporciona los certificados SSL utilizados para HTTPS en producción. Esto garantiza una comunicación privada entre cliente y servidor. Let's Encrypt es una autoridad gratuita de certificados que los gestiona mediante Certbot de manera

automática. Certbot permite realizar un tedioso proceso de obtención de certificados y firmas de autoridad mediante un par de comandos, agilizando mucho la implementación de HTTPS.

PM2 se ocupa de mantener la aplicación de Node.js ejecutándose, gestiona reinicios automáticos y además monitoriza su estado. Se puede encontrar información sobre su configuración en el Anexo I

4.1.2.2 Clientes y flujo de interacción

El usuario entrará a www.chischas.xyz desde cualquier navegador web y gracias a la resolución de dominio de Namecheap hacia la IP elástica de la instancia EC2 este accederá sin molestia. La interacción de tiempo real con el rival se establece gracias a WebSockets, permitiendo partidas en tiempo real, historial de movimientos y chat en vivo. La comunicación entre el servidor de aplicación y la base de datos MongoDB se realiza de manera local dentro de la instancia, esto reduce la superficie de exposición haciéndolo más seguro y optimizando el rendimiento.

Para la implementación de la seguridad y las reglas de acceso comenzaremos con los detalles del grupo de seguridad de la EC2. El puerto SSH está reservado para que los administradores puedan conectarse a la EC2. El tráfico web está permitido a través del puerto HTTP y HTTPS. En la captura (ilustración 9 Captura de pantalla interfaz AWS) el puerto 3000 está abierto temporalmente por desarrollo (debería protegerse). El puerto de la MongoDB (27017) está limitado para que solo la propia instancia pueda acceder.

Las reglas de salida no tienen seguridad alguna, permiten todo el tráfico para la instalación de actualizaciones y dependencias, como puede ser un git pull de una nueva versión de “Chischás!” ya que primero se establece una conexión mediante nuestra solicitud de pull, lo que permite la descarga del repositorio. Sin esa conexión inicial, nadie podría inyectarnos datos al servidor, es el git pull el que valida la entrada.

Ilustración 9 Captura de pantalla de interfaz de seguridad AWS

sg-Oda433f9159f3aedb - Chiskas_seguridad Acciones

Detalles
Nombre del grupo de seguridad
Chiskas_seguridad
Propietario
710560916670

ID del grupo de seguridad
sg-Oda433f9159f3aedb
Número de reglas de entrada
5 Entradas de permisos

Descripción
Default
Número de reglas de salida
1 Entrada de permiso

ID de la VPC
vpc-067c6994570aab608

Reglas de entrada | Reglas de salida | Compartiendo : *novedad* | Asociaciones de VPC : *novedad* | Etiquetas

Reglas de entrada (5) Administrar etiquetas Editar reglas de entrada

<input type="checkbox"/>	Name	ID de la regla del gr...	Versión de IP	Tipo	Protocolo	Intervalo de puertos	Origen	Descripción
<input type="checkbox"/>	-	sgr-0991fd8610f33671b	IPv4	TCP personalizado	TCP	3000	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-06f6bc37f87c3fa88	IPv4	SSH	TCP	22	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-0788d339b6a15cdfd	IPv4	TCP personalizado	TCP	27017	127.0.0.1/32	-
<input type="checkbox"/>	-	sgr-001e91090805a67ac	IPv4	HTTP	TCP	80	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-09ca27dd722f8458f	IPv4	HTTPS	TCP	443	0.0.0.0/0	-

Nota. los grupos de seguridad permiten establecer configuraciones que luego se pueden aplicar a diferentes instancias sin duplicar parámetros. Tomado de (propia, Vista del grupo de seguridad “Chiskas_seguridad” en AWS EC2, 2025)

Gestión del dominio y DNS

Ilustración 10 Dirección IP elástica

Direcciones IP elásticas (1/1) Información Acciones Asignar dirección IP elástica




Buscar direcciones IP elásticas por atributo o etiqueta

<input checked="" type="checkbox"/>	Name	Dirección IPv4 asign...	Tipo	ID de asignación	Registro DNS inverso	ID de instancia asociada	Dirección IP privada
<input checked="" type="checkbox"/>	-	34.199.73.164	IP pública	eipalloc-008ffc7382cf6e94f	-	i-0c98568ada0b3e3f5	172.31.92.217

Nota. esta dirección supone un coste extra, pero es necesaria, ya que permite que la IP a la que apunta el DNS sea siempre la misma tras un reinicio de la EC2. Tomado por (propia, Vista de las Direcciones IP elásticas en AWS EC2, 2024)

El DNS está gestionado mediante Namecheap. Namecheap también es la plataforma gestora del dominio. Si nos adentramos a la configuración del dominio “chiskas.xyz” tenemos rutas apuntando a la IP elástica de nuestra EC2, esto le da una apariencia profesional a la web además de mucha sencillez, ya que es lo que hace posible el funcionamiento del dominio.

Ilustración 11 Direcciones DNS Namecheap

<input type="checkbox"/>	Type	Host	Value	TTL	
<input type="checkbox"/>	A Record	@	34.199.73.164	Automatic	
<input type="checkbox"/>	A Record	www	34.199.73.164	Automatic	
<div>  ADD NEW RECORD </div>					

Nota. Namecheap tiene su propia interfaz en línea. Tomado por (propia, Configuración de registros A en Namecheap, 2025)

El despliegue sigue un modelo monolítico de tres capas: servidor, base de datos y proxy inverso (NGINX) dentro de un entorno monitorizado y seguro. Consta con medidas de persistencia como el software PM2, cifrado HTTPS con Let's Encrypt y políticas de grupos de seguridad que garantizan la integridad y disponibilidad del servicio. Todo esto se ha explicado con detalle anteriormente. El candado de la ilustración 12 es la manera de verificar si HTTPS funciona.

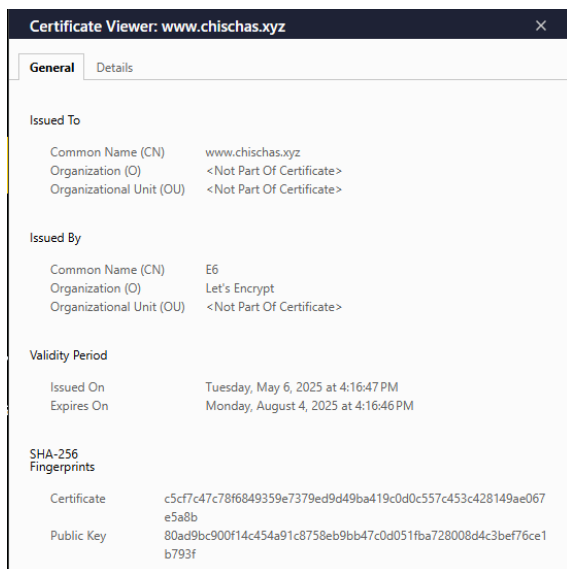
Ilustración 12 Candado Https en navegador



Nota. Una vez HTTPS está funcionando correctamente, se muestra en el navegador el candado previamente mencionado, indicando que la conexión es segura. Esto simplemente verifica que la conexión con www.chischas.xyz está cifrada. Captura de pantalla de navegador Opera.

La ilustración 13 muestra que pasa si haces clic en el candado del navegador. Esta acción nos muestra cuando se firmó el certificado, cuando caduca y quien lo ha firmado, entre otros factores.

Ilustración 13 Certificado digital de www.chischas.xyz



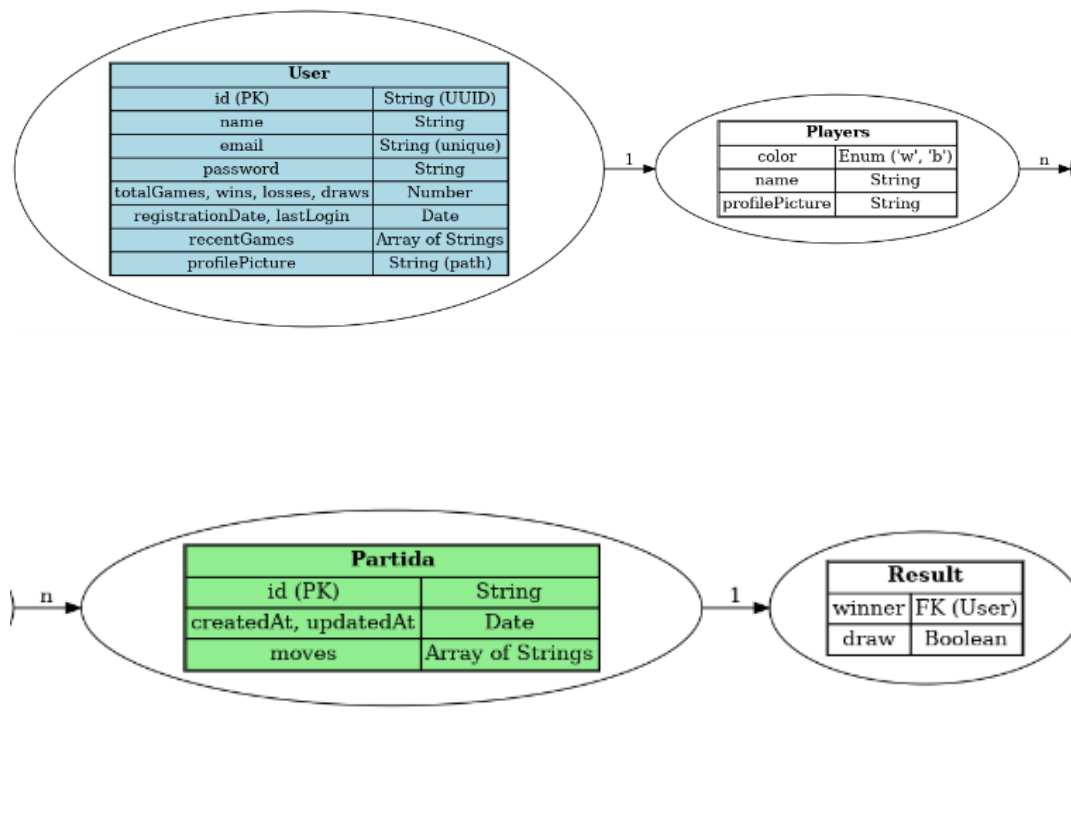
Nota. esta captura muestra como el certificado que permite la conexión cifrada con el servidor de “Chischás!” está firmado por la organización “Let’s Encrypt”. Captura de pantalla de navegador Opera.

Es una solución robusta para un futuro despliegue a mayor escala debido a la simpleza que propone.

4.2 Diseño de datos

El diseño de datos está basado en MongoDB, por lo cual, es un diseño no relacional como se ve en la ilustración 14, a pesar de que las aparentes relaciones son entidades más pequeñas almacenadas por la anterior entidad (más grande). Esto no quiere decir que no haya “relaciones” entre datos, si no que no hay claves foráneas ni JOINS como en SQL.

Ilustración 14 Diagrama de diseño de datos



Nota. El diagrama muestra el flujo de los datos dentro del MongoDB de “Chischás!”.

Elaboración propia

Las relaciones son tres:

1. Un User puede jugar en muchas Partidas (relación n a n gestionada por Players).
2. Cada Partida tiene un único Result (relación 1 a 1).
3. Players define cómo participa un User en cada Partida.

Consta de dos únicas colecciones: Partidas y usuarios (users).

User es la entidad principal que representa a cada usuario. Su clave primaria es id. Tiene atributos como name, email (único), password, estadísticas (totalGames, wins, losses, draws), fechas (registrationDate, lastLogin), un array de partidas recientes (recentGames) y la ruta de la foto de perfil (profilePicture). A continuación, un JSON extraído de la BDD .

```
{
  "_id": {
    "$oid": "67fe974e4604a6bf7f5bd76a"
  },
  "name": "Elisabeth",
  "email": "elisabeth@gmail.com",
  "password":
"$2b$10$Jn.exvSkqoBCm8YCnEhUdeeS0mdmQmgCaF.vfVdaRWiS7p/sA58B.",
  "totalGames": 5,
  "wins": 4,
  "losses": 1,
  "draws": 0,
  "recentGames": [],
  "profilePicture": "/uploads/67fe974e4604a6bf7f5bd76a-
1747131263274.png",
  "id": "bc626abb-41ef-4f96-8497-f5469e65f1ab",
  "registrationDate": {
    "$date": "2025-04-15T17:28:46.624Z"
  },
  "__v": 0
}
```

Players es la relación que conecta usuarios con partidas. No es una entidad en sí, sino una relación débil que guarda información de contexto: color (w/b), name (en la partida) y profilePicture. Permite que un usuario esté en varias partidas y cada partida tenga sus jugadores.

Partida es la entidad que almacena cada partida de ajedrez, con id (PK) como identificador. Guarda también las fechas de creación y actualización (createdAt, updatedAt) y el listado de movimientos (moves).

Result es una subentidad de Partida que define el resultado de la partida. Tiene un atributo winner y draw (booleano).

Así se vería una partida en la base de datos.

```
{
  "_id": {
    "$oid": "6818bf66bbb76677375012f6"
  },
  "id": "b07179ad-5986-4a7f-8f8f-61ca65c530cc",
  "players": [
    {
      "userId": {
        "$oid": "6818bf26bbb76677375012e7"
      },
      "color": "w",
      "name": "juan",
      "profilePicture": "/uploads/default-profile.jpg",
      "_id": {
        "$oid": "6818bf66bbb76677375012f7"
      }
    },
    {
      "userId": {
        "$oid": "6818bef2bbb76677375012db"
      },
      "color": "b",
      "name": "pedro",
      "profilePicture": "/uploads/6818bef2bbb76677375012db-1746452223104.png",
      "_id": {
        "$oid": "6818bf66bbb76677375012f8"
      }
    }
  ],
  "result": {
    "winner": {
      "$oid": "6818bf26bbb76677375012e7"
    },
    "draw": false
  },
  "moves": [
    "e4",
    "e5",
    "Nf3",
    "a6",
    "Nh4",
    "a5",
    "Qf3",
    "a4",
    "Bc4",
    "a3",
    "Qf6",
    "b5",
    "Qxf7#"
  ],
  "createdAt": {
    "$date": "2025-05-05T13:38:46.589Z"
  },
  "updatedAt": {
    "$date": "2025-05-05T13:38:46.589Z"
  }
}
```



```

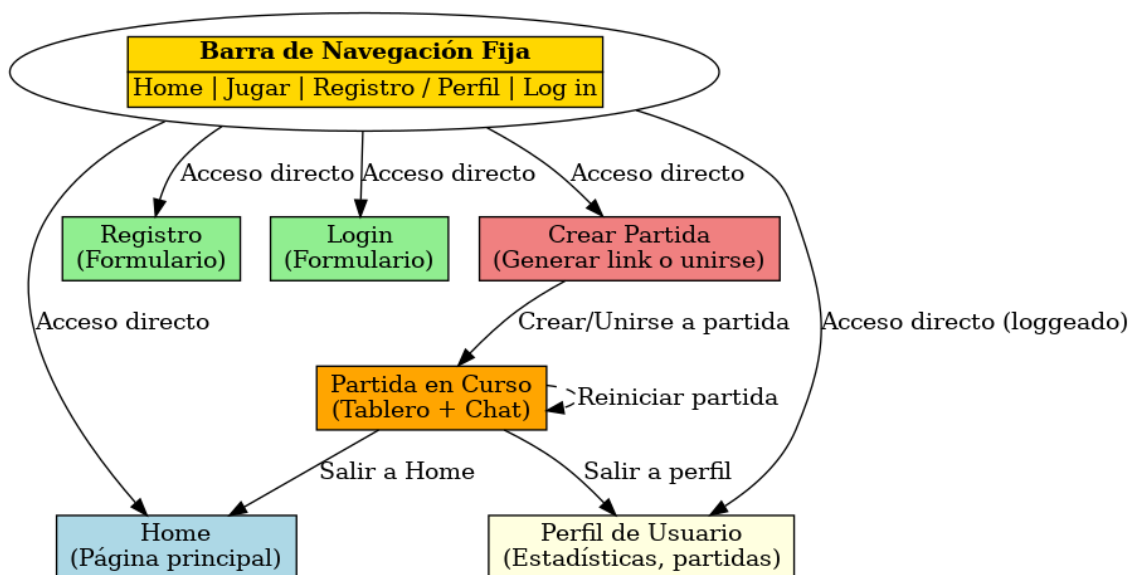
},
"__v": 0
}

```

4.3 Interfaz de usuario

La interfaz de Chischás fue pensada para utilizarse con una barra de navegación superior fija permanente en todas las diferentes pantallas. La ilustración 15 lo representa.

Ilustración 15 Diagrama del diseño de la interfaz de usuario



Nota. Elaboración propia

Esta barra tendrá los accesos directos a home (el índice), jugar, registro o perfil (depende si ya se ha iniciado la sesión o no) y el inicio de sesión (el cual desaparece si la sesión está iniciada).

El home de la página muestra información de la web, el enlace al repositorio y como reportar errores.

Ilustración 16 Home de Chischás!



Jugar permite la creación de un enlace de juego y comenzar la partida. Se pueden jugar tantas partidas a la vez como el usuario quiera y tantos rivales como tenga.

Ilustración 17 Menú de juego de Chischás!



La ilustración 18 muestra el diseño de una partida. Se puede ver todas las funcionalidades que se explicarán en el apartado 4.4.

Ilustración 18 Partida en línea de Chischás!



El registro contiene un simple formulario para la creación de la cuenta. Se perderá el acceso directo a este formulario una vez el usuario inicie la sesión, transformándose en el acceso directo al perfil.

Ilustración 19 Formulario de registro de Chischás!



El inicio de sesión contiene el formulario donde se introducen los datos y son enviados al servidor para compararlos con los registros de la BDD. Una vez se inicie sesión se pierde el acceso a esta ruta.

Ilustración 20 Formulario de Login de Chischás!



La interfaz de perfil permite modificar el nombre de usuario, mail o foto de perfil. Además, enseña un recuento de las partidas totales, ganadas, perdidas y empatadas junto a la fecha de registro inicial del usuario y un historial de partidas jugadas. El historial contiene la fecha de cada partida, el rival, resultado y el color con el que jugaba. Finalmente, hay un botón de cerrar sesión que además elimina la cookie para que si vuelves a entrar en la web la sesión no se mantenga.

Ilustración 21 Perfil de usuario de Chischás! Ejemplo Elisabeth (perfil invitado)

Chischás!
by Miguel Gamboa Sánchez

Home Jugar Elisabeth

Perfil de Usuario

Nombre: Elisabeth
Email: elisabeth@gmail.com
Foto de perfil: No file chosen

Partidas Totales: 5
Partidas Ganadas: 4
Partidas Perdidas: 1
Partidas Empatadas: 0
Fecha de Registro: 15/04/2025

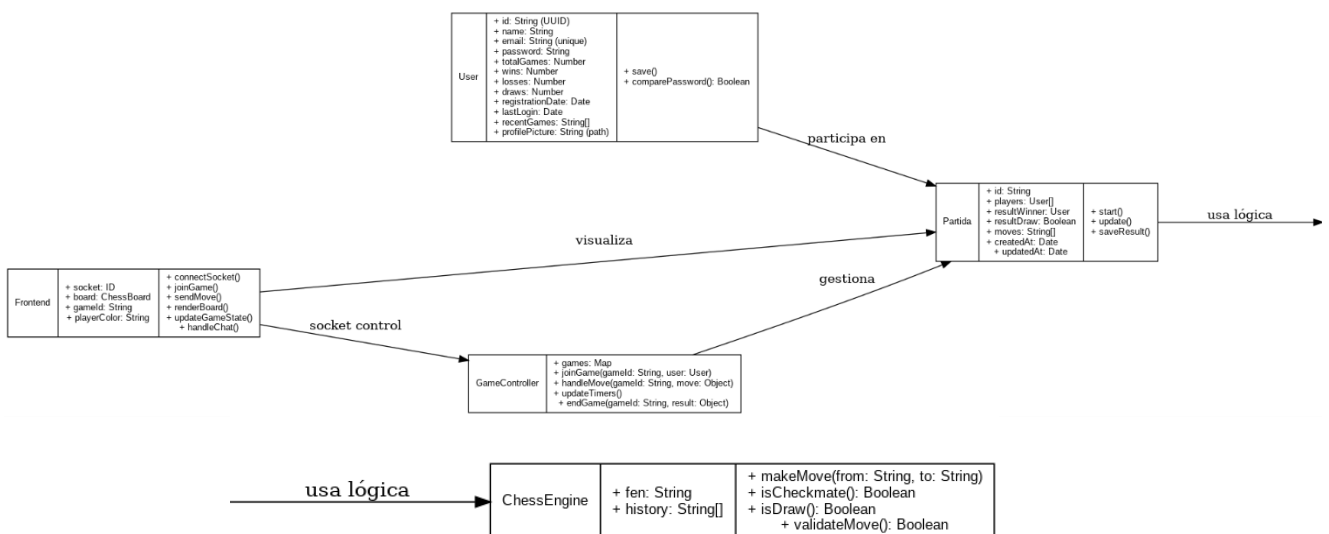
Partidas Jugadas

Fecha	Rival	Resultado	Color
24/04/2025	a	Derrota	Negras
24/04/2025	a	Victoria	Bancas
24/04/2025	a	Victoria	Bancas
24/04/2025	a	Victoria	Bancas
25/04/2025	hehehehehehehehe	Victoria	Bancas
25/04/2025	hehehehehehehehe	Derrota	Negras
01/05/2025	manin	Victoria	Bancas

4.4 Diagrama de clases

“Chischás!” arranca un Express habilitando las rutas junto a las vistas EJS. Levanta un servidor HTTP junto con la creación de sockets preparados para el comienzo de partidas mediando Socket.io. Carga los modelos de usuarios y partidas, además de dar inicio al motor de juego (chess.js). También configura las sesiones (express sessions) para que los usuarios tengan su login persistente durante 7 días.

Ilustración 22 Diagrama de clases de la aplicación



Nota. el diagrama enseña cómo funciona toda la lógica de programación para que Chischás! pueda realizar todas sus funcionalidades. Elaboración propia

Cuando un jugador accede a la partida, establece una conexión vía socket con el servidor, el cual le irá informando de todos los sucesos, además de enviar una señal cada segundo para que su reloj tenga tiempo. El frontend carga datos como el `userId` o el nombre del jugador desde la etiqueta `body` y envía el evento `joinGame` con el `id` de partida que contiene el enlace de acceso. Si un jugador no ha iniciado sesión se ven en la obligación de rellenar un cuestionario (prompt) con un nombre deseado.

Cuando entra `joinGame` en el backend, comprueba si la partida existe en el objeto `games`, si no existe la crea. Cada jugador es asignado con su `socketId`, nombre, foto de perfil y `userId` en caso de tener una cuenta, será `null` en caso contrario. Con esto el servidor puede identificar a cada jugador permitiendo el sorteo del color (evento `colorAssignment`) y la asignación de un lado del tablero. Cuando ya están los dos jugadores unidos y esto se ha realizado, se lanza el evento `startCountdown`. Este da comienzo a una animación de Versus con una cuenta atrás de 3 segundos, enseñando a los jugadores sus respectivos colores. “La partida comienza en 3, 2, 1...”. Esto permite sincronizar a los dos jugadores para que empiecen la partida al mismo tiempo. Además, el color de cada usuario determina la orientación del tablero.

Cuando la partida comienza, se inicializa el tablero gracias a `Chessboard.js` (permite visualizar) y `Chess.js` (soporta la lógica y no deja ver el tablero hasta que esta esté funcionando para evitar trampas).

`Chess.js` valida si cada movimiento es legal, si no lo es, devuelve la pieza a su sitio. En cambio, si es válido, se envía el movimiento por sockets y el servidor lo valida. Una vez la señal ya está en el backend, se añade el movimiento al historial y se actualiza el FEN (es la notación que describe el estado del tablero).

El sistema de notación Forsyth-Edwards (FEN) es un tipo de notación utilizado en ajedrez. Propuesto en 1883, a diferencia de los sistemas de notación para jugadas y partidas, este sistema se utiliza para anotar una posición.

(Wikipedia Contributors, 2025)

Además, en cada movimiento comprueba si la partida ha terminado.

Si la partida termina, el backend guarda el resultado en la BDD dentro de la colección partidas y si los dos usuarios participantes están registrados se actualizan sus estadísticas. Después, el backend borra el objeto de la partida para optimizar el funcionamiento si hubiese muchas partidas.

El frontend durante la partida estuvo recibiendo continuamente los eventos `opponentMove`, `gameState` y `updateTimer`, permitiendo el actualizado de tablero, historial de movimientos y el reloj de cada jugador. Ambos jugadores ven en tiempo real el reloj del otro y es el jugador que recibe señales vía socket aquel que está perdiendo segundos, así que cuanto antes mueva antes le pasará el chorro al rival. El primero en quedarse sin puntos (o segundos) perderá la partida como si fuese un jaque mate.

`ChatMessage` es otro evento que permite hacer rebotar mediante el servidor al otro usuario el mensaje que quiera, permitiendo la comunicación entre jugadores. En un futuro se podría implementar en el servidor un filtrado de lenguaje o un traductor, para hacer el chat más consistente.

La lógica de juego nunca la maneja el frontend, permitiendo un sistema de juego seguro ya que toda jugada la valida `Chess.js`, y el backend lo almacena. Esto evita manipulaciones del JS en local permitiendo mates instantáneos o muchas más cosas.

Como se puede ver, “Chischás!” alberga una lógica rápida y sencilla a sus espaldas, se pueden encontrar más detalles sobre las librerías instaladas en el Anexo II.

4.5 Entorno de construcción

Para poder llevar todo esto a un buen puerto ha sido necesario el uso de diferentes herramientas de desarrollo:

- a. Visual Studio Code: IDE para la edición de código, visualización de la BDD, gestión de dependencias y ejecución de scripts. La BDD pudo ser visualizada con más comodidad gracias a una extensión llamada “MongoDB for VS Code”.
- b. Github: para la gestión del repositorio y versionado de “Chischás!”. Además, partes muy puntuales del código fueron asistidas mediante recomendaciones instantáneas proporcionadas por GitHub Copilot. Todas las sugerencias fueron revisadas y adaptadas a mano para asegurar su adecuación al proyecto.
- c. Navegadores Opera y Google Chrome: se utilizaron dos navegadores para la realización de pruebas de partidas entre usuarios con diferentes cuentas, esto se debe a que las sesiones daban problemas ya que el propio navegador las guarda y son varios los usuarios que queríamos probar. Además, fueron imprescindibles para el diseño de interfaz.
- d. Servidor local: la aplicación fue ejecutada en localhost:3000 antes de la etapa de producción, y aun así, la propia instancia EC2 ejecuta la aplicación en su localhost también, pero, como ya se ha explicado, gracias a NGINX hace el traspaso de HTTPS al puerto 3000.
- e. MongoDB local: base de datos no relacional ejecutada en el entorno local para el almacenamiento de los datos ya explicados y la implementación de la misma en la aplicación mediante Mongoose.

Este apartado, realmente va a gusto de cada desarrollador, el uso de otras herramientas no tiene porque afectar en el funcionamiento de “Chischás!”.

4.6 Referencia al repositorio de software

Repositorio:

<https://github.com/MiguelG7/Chischas.git>

Web:

<https://www.chischas.xyz> (debido al plan educativo de AWS proporcionado por la universidad, la web de “Chischás!” se cae cada 4 horas, por lo cual, si se desea hacer la prueba se debe levantar primero).

Capítulo 5

Validación del sistema

5.1 Plan de pruebas

Con el objetivo de evaluar el rendimiento de “Chischás!” en su capacidad de respuesta ante consultas y la aparición acelerada de usuarios en la web, se ha diseñado la siguiente prueba sustentada por Artillery.io

Artillery.io es un kit de herramientas de código abierto que facilita las pruebas de rendimiento debido a su gran utilidad en pruebas de carga mediante scripts en YAML y su interfaz de seguimiento y resultados. La prueba realizada generó el siguiente informe, a continuación, será explicada con más detalle. Si lo desea, puede ver la prueba con más detalle en el siguiente link (las imágenes, han sido extraídas de ahí):

https://app.artillery.io/share/sh_5a5d9690a1101f0c6be12f91f90fdaf0218a7047504a6725452d222ab82c4099

Prueba de carga de usuarios (RAMP UP) utilizando Artillery.io.

Metodología:

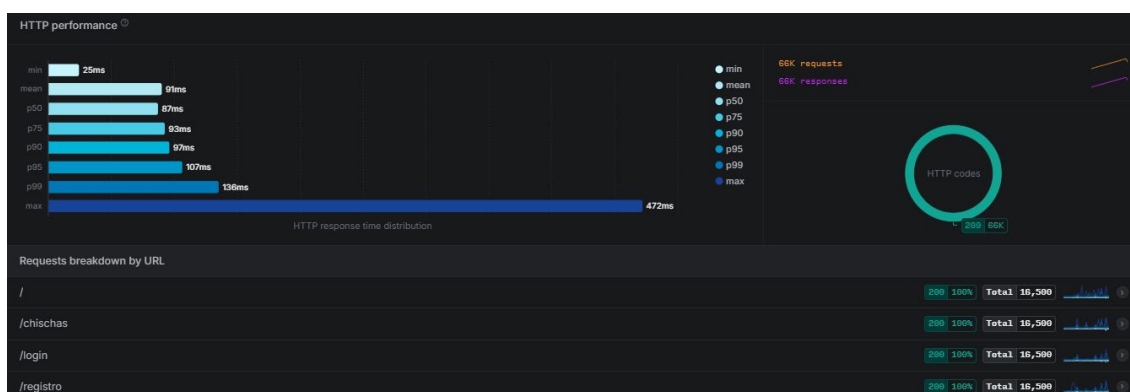
Se realiza una prueba de carga escalonada mediante un script YAML conectado con Artillery que aumenta la carga de usuarios progresivamente en un tiempo de 5 minutos. Las ilustraciones 23 y 24 muestran el escalado de carga de solicitudes ya que comienza con 10 usuarios por segundo y finaliza con 100.

Ilustración 23 Resumen de prueba de carga



Nota. tomado de (Artillery, Reporte test de carga: 16,500 usuarios virtuales, 2025)

Ilustración 24 Informe de rendimiento HTTP: tiempos de respuesta y desglose por endpoint



Nota. Tomado de (Artillery, Informe de rendimiento HTTP: tiempos de respuesta y desglose por endpoint, 2025)

Pruebas de carga: Capacidad máxima de usuarios al mismo tiempo en:

- Página principal (/)
- /chisclas
- /registro
- /login

Se representaron los datos en formato tabla para su fácil entendimiento en la tabla 9.

Tabla 9 Resultados de prueba de carga

Métrica	Resultado	Interpretación
Total de usuarios simulados	16500	Usuarios virtuales ejecutando el flujo de navegación.
Total de peticiones realizadas	66000	Incluye todas las rutas accedidas.
Peticiones por segundo (sostenidas)	217 req/s	Capacidad de respuesta del servidor bajo carga.
Tiempos de respuesta promedio	90,6 ms	Excelente rendimiento (<100 ms).
Tiempo de respuesta p95	106,7 ms	El 95% de las peticiones responden en menos de 107 ms.
Tiempo de respuesta p99	135,7 ms	El 99% responden en menos de 136 ms.
Errores (fallos de conexión o respuesta)	0	Ningún fallo detectado.

Como se puede apreciar, “Chischás!” puede aguantar hasta 16500 usuarios en la página y hasta una media 217 peticiones simultaneas por segundo, llegando a un pico de 386. Con los datos del análisis hemos realizado un cálculo estimado de cuantos jugadores podrían estar en partida simultáneamente y se muestra en la tabla 10:

Tabla 10 Estimación máximos usuarios en partida

Parámetro	Valor	Unidad / Observaciones
Mensajes por jugada (entrada + salida)	2	mensajes
Frecuencia de jugadas en partidas rápidas (blitz)	1 jugada cada 5 segundos	0.2 jugadas/s
Mensajes por segundo por partida	0.4	mensajes/s
Capacidad estimada del servidor en mensajes WebSocket/s	2500	mensajes/s (conexiones ligeras)
Partidas activas simultáneas soportadas	6250	partidas simultáneas en tiempo real
Jugadores jugando simultáneamente (2 por partida)	12500	jugadores jugando activamente
Usuarios conectados navegando (HTTP concurrentes)	16500	usuarios navegando simultáneamente (HTTP requests)

La interpretación de estos resultados está bastante clara en con las tablas y esto nos permite afirmar que validan el cumplimiento de los requisitos no funcionales del apartado 3.1.

Capítulo 6

Conclusiones y líneas futuras

Ideas a las que se llega después del desarrollo del proyecto, así como las líneas posibles de trabajo posterior.

6.1 Conclusiones

El desarrollo de esta web me ha permitido darme cuenta de las dimensiones reales de un proyecto así, haciéndome asimilar que si quisiese hacer un proyecto mucho más grande y ambicioso sería importante conseguir un buen equipo de trabajo, ya que uno solo nunca podrá llegar a lo que hace un buen equipo.

Comencé con la idea de realizar yo mismo el motor del ajedrez, pero a medida que aumentaban las funcionalidades que debía implementar me iba dando cuenta de lo extenso que era el proyecto, a pesar de que en un inicio pareciese la cosa más simple del mundo.

Así que con esto también concluí que los proyectos vistos desde fuera parecen mucho más simples de lo que uno cree y a veces la simplicidad es el peor enemigo de un desarrollador.

Aun así, “Chischás!” cumple los objetivos planteados, ofreciendo una solución funcional y ampliable, que permite a cualquier usuario jugar partidas de ajedrez en línea de manera rápida, directa y sin complicaciones.

6.2 Líneas futuras

Una vez finalizado el desarrollo actual, surgen diversas líneas de trabajo que permitirían ampliar y mejorar la aplicación. Funcionalidades como la reconexión en partidas, el sistema de emparejamiento automático, el historial público y análisis de partidas, modo espectador en vivo, optimización de la infraestructura, integración de sistema de ELO y ranking, aplicación móvil... Se podría hacer de “Chischás!” una plataforma de ajedrez más enfocada al mundo competitivo, que al final es lo que buscan la mayoría de los jugadores. Por suerte, al ser código abierto, la posibilidad queda abierta para futuros desarrolladores.

Capítulo 7 Bibliografía

Agencia Española de Protección de Datos. (junio de 2025). *¿Qué es el principio de responsabilidad proactiva?* Obtenido de Agencia Española de Protección de Datos: <https://www.aepd.es/preguntas-frecuentes/2-rgpd/3-principios-relativos-al-tratamiento/FAQ-0208-que-es-el-principio-de-responsabilidad-proactiva>

Artillery. (20 de mayo de 2025). *Informe de rendimiento HTTP: tiempos de respuesta y desglose por endpoint.* Obtenido de Artillery: https://app.artillery.io/share/sh_5a5d9690a1101f0c6be12f91f90fdaf0218a7047504a6725452d222ab82c4099

Artillery. (20 de Mayo de 2025). *Reporte test de carga: 16,500 usuarios virtuales.* Obtenido de Artillery: https://app.artillery.io/share/sh_5a5d9690a1101f0c6be12f91f90fdaf0218a7047504a6725452d222ab82c4099

Automattic. (s.f.). Obtenido de Mongoose: Elegant MongoDB object modeling for Node.js: <https://mongoosejs.com>

Broofa. (s.f.). *uuid*. Obtenido de npm: <https://www.npmjs.com/package/uuid>

Comparasoftware. (2025). *Modelo incremental: ¿Qué es y cuáles son sus fases?* Obtenido de Comparasoftware Blog: <https://blog.comparasoftware.com/wp-content/uploads/2021/06/fases-del-mod-incremental.png>

Digital, M. d. (s.f.). *Métrica v3: Metodología para el desarrollo de sistemas de información.* Obtenido de Portal de Administración Electrónica:

https://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html

El País. (Junio de 2025). *Jaque mate al coronavirus: el imparable crecimiento del ajedrez online durante la pandemia*. Obtenido de El País: <https://elpais.com/tecnologia/2020-06-06/jaque-mate-al-coronavirus-el-imparable-crecimiento-del-ajedrez-online-durante-la-pandemia.html>

Embedded JavaScript templates. (s.f.). Obtenido de EJS: Embedded JavaScript templates: <https://ejs.co/>

Express.js. (s.f.). Obtenido de Express – Node.js web application framework: <https://expressjs.com/>

Express.js. (s.f.). Obtenido de Express-session: <https://expressjs.com/en/resources/middleware/session.html>

jhlywa. (s.f.). *chess.js (v0.13.0)*. Obtenido de npm: <https://www.npmjs.com/package/chess.js/v/0.13.0>

Kundu, K. (s.f.). *npm*. Obtenido de multer: <https://www.npmjs.com/package/multer>

Microsoft. (2025). *Creación de un proxy directo usando Application Request Routing*. Obtenido de Microsoft Learn: <https://learn.microsoft.com/es-es/iis/extensions/configuring-application-request-routing-arr/creating-a-forward-proxy-using-application-request-routing>

Motdotla. (s.f.). *dotenv*. Obtenido de npm: <https://www.npmjs.com/package/dotenv>

Naciones Unidas. (Junio de 2025). *Día Mundial del Ajedrez*. Obtenido de Naciones Unidas.

Naciones Unidas. (junio de 2025). *Día Mundial del Ajedrez*. Obtenido de Naciones Unidas: <https://www.un.org/es/observances/world-chess-day>

Oakmac. (s.f.). *chessboard.js*. Obtenido de GitHub:
<https://github.com/oakmac/chessboardjs>

propia, C. d. (Mayo de 2024). *Vista de la región del servidor en AWS*. Obtenido de Amazon Web Services.

propia, C. d. (2024). *Vista de las Direcciones IP elásticas en AWS EC2*. Obtenido de Amazon Web Services.

propia, C. d. (Mayo de 2025). *Configuración de registros A en Namecheap*. Obtenido de Namecheap: <https://www.namecheap.com>

propia, C. d. (Mayo de 2025). *Vista del grupo de seguridad "Chischas_seguridad" en AWS EC2*. Obtenido de Amazon Web Services.

Sánchez, M. G. (Mayo de 2025). *Package.json "Chischás"*. Obtenido de Repositorio Github "Chischás!":
<https://github.com/MiguelG7/Chischas/blob/main/package.json>

Socket.IO. (s.f.). Obtenido de Socket.IO documentation (v4): <https://socket.io/docs/v4/>

Wikipedia Contributors. (2025). *Notación de Forsyth-Edwards*. Obtenido de Wikipedia:
https://es.wikipedia.org/wiki/Notación_de_Forsyth-Edwards

Bibliografía citada en la memoria. Todas las referencias deben seguir un **formato homogéneo**. [Formato APA](#).

Anexo I - Configuración de NGINX y HTTPS:

Como ya hemos comentado antes NGINX actúa como proxy inverso dentro de la EC2. Este recibe peticiones en el puerto 80 HTTP y las redirige al puerto 3000 que es donde está trabajando Node.js.

Una vez esto funciona, configuramos Let's Encrypt para que la conexión al puerto 80 se redirija al 443 donde está ya el servicio HTTPS. Además debemos activar en nuestra node el HTTPS TRUE ONLY para que solo permita conexiones mediante la SSL cifrada.

Configuración de NGINX:

```
server {  
    listen 80;  
    server_name www.chischas.xyz chischas.xyz;  
  
    location / {  
        proxy_pass http://localhost:3000;  
        proxy_http_version 1.1;  
        proxy_set_header Upgrade $http_upgrade;  
        proxy_set_header Connection 'upgrade';  
        proxy_set_header Host $host;  
        proxy_cache_bypass $http_upgrade;  
    }  
}
```

Acepta las conexiones al puerto 80 de www.chischas.xyz y chischas.xyz mandándolas al localhost:3000 que estará siempre funcionando gracias a la PM2. Existe la misma configuración en la 443. Aun así, el navegador, por defecto redirigirá las peticiones al puerto 443 (HTTPS)

Gestión de procesos con PM2.

Comandos realizados en la consola de la Ubuntu tras instalar PM2

1. Iniciamos la aplicación: `pm2 start app.js`
2. Guardamos la configuración inicial: `pm2 save`
3. Configuramos en arranque automático: `pm2 startup`

En una de las fases del despliegue, se solventó el error de intentar ejecutar “pm2 start chischas” (que falló porque no era un archivo), corrigiéndolo con:

```
cd ~/Chischas  
pm2 start app.js --name chischas
```

Posteriormente, se configuró el arranque persistente tras reinicios con:

```
sudo env PATH=$PATH:/usr/bin /usr/lib/node_modules/pm2/bin/pm2 startup  
systemd -u ubuntu --hp /home/ubuntu  
pm2 save
```

Anexo II – Librerías utilizadas

Comenzaremos explicando las librerías desde el documento que las contiene (package.json), además finalmente, los motores de la lógica de juego.

package.json:

```
{
  "dependencies": {
    "cookie-parser": "^1.4.7",
    "dotenv": "^16.4.7",
    "ejs": "^3.1.10",
    "express": "^4.21.2",
    "express-session": "^1.18.1",
    "mongoose": "^1.0.0",
    "multer": "^1.4.5-lts.2",
    "socket.io": "^4.8.1",
    "socket.io-client": "^4.8.1",
    "uuid": "^11.1.0"
  }
}
```

Nota. captura sacada del repositorio. Tomada por (Sánchez, 2025)

Express es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles.

(Express.js, s.f.).

EJS es un lenguaje de plantillas sencillo que permite generar marcado HTML con JavaScript simple. (“EJS - Plantillas JavaScript Incrustadas”) Sin formalismos en la organización. Sin reinventar la iteración ni el flujo de control. Es simplemente JavaScript simple.

(Embedded JavaScript templates, s.f.)

Express-session: Proporciona una función que devuelva una cadena que se utilizará como ID de sesión.

Nota: Desde la versión 1.5.0, ya no es necesario usar el middleware del analizador de cookies para que este módulo funcione. Este módulo ahora lee y escribe cookies directamente en la solicitud/respuesta. (“node.js - node - passport js - edu.lat -

tutoriales.edu.lat”) El uso del analizador de cookies puede causar problemas si el secreto no es el mismo entre este módulo y el analizador de cookies.

(Express.js, s.f.)

Dotenv es un módulo sin dependencias que carga variables de entorno desde un archivo .env a process.env. El almacenamiento de la configuración en el entorno, separado del código, se basa en la metodología de la aplicación de doce factores.

(“EniDev | Guías sobre temas de bases de datos y programación”)

(Motdotla, s.f.)

Mongoose ofrece una solución sencilla y basada en esquemas para modelar los datos de su aplicación. Incluye conversión de tipos, validación, creación de consultas, enlaces de lógica de negocio y mucho más, todo listo para usar.

(Automattic, s.f.)

Multer es un middleware de Node.js para gestionar datos multipart/form, utilizado principalmente para subir archivos. Está desarrollado sobre busboy para una máxima eficiencia.

(Kundu, s.f.)

Nota. utilizado en las fotos de perfil de los usuarios de Chischás!

Socket.IO es una biblioteca que permite la comunicación bidireccional, de baja latencia y basada en eventos entre un cliente y un servidor. (“GLAB S06 Rcoello 2024 01 1 - Carrera: Diseño y Desarrollo de ... - Studocu”)

(Socket.IO, s.f.)

El ID de nodo predeterminado (los últimos 12 dígitos del UUID) se genera una vez, de manera aleatoria, al iniciar el proceso y luego permanece sin cambios mientras dura el proceso.

(Broofa, s.f.)

Nota. utilizado para crear los enlaces de partidas de Chischás!

Motor de ajedrez y visualización

El núcleo funcional de juego se apoya en dos librerías de código abierto

chess.js es una biblioteca de ajedrez JavaScript que se utiliza para la generación/validación de movimientos de ajedrez, la colocación/movimiento de piezas y la detección de jaque/jaque mate/estancamiento: básicamente todo excepto la IA.
(jhlywa, s.f.)

chessboard.js es un tablero de ajedrez JavaScript independiente. Está diseñado para ser "simplemente un tablero" y ofrece una potente API para que pueda usarse de diversas maneras. (Oakmac, s.f.)

Ambas librerías son utilizadas respetando sus licencias de código abierto disponibles en los repositorios (enlace bibliografía).

Glosario de términos

Ordenado alfabéticamente:

AWS (Amazon Web Services): Plataforma en la nube donde está “Chischás!” desplegada.

BDD (Base de datos): Software lógico donde se almacenan los datos.

CSS (Cascading Style Sheets): Lenguaje que permite trabajar la parte visual de sistema web.

DoS (Denial of Service): Ataque informático que busca saturar un servidor para que deje de responder.

EC2 (Elastic Compute Cloud): Ordenador virtual en la nube de AWS el cual es accesible por SSH y se mantiene en pie.

FEN (Forsyth-Edwards Notation): Notación usada para describir la posición de las piezas de en un instante de una partida de ajedrez

HTML (HyperText Markup Language): Lenguaje común en la organización del contenido de páginas web.

HTTP (HyperText Transfer Protocol): Protocolo utilizado para enviar información a servidores web, normalmente HTML.

HTTPS (HyperText Transfer Protocol Secure): Protocolo que cifra la conexión HTTP entre cliente y servidor.

IP (Internet Protocol): Número que identifica un dispositivo en una red.

JS (JavaScript): Lenguaje que permite programar la lógica del sistema web.

LOPDGDD (Ley Orgánica de Protección de Datos y Garantía de Derechos Digitales): Ley española que adapta el RGPD y lo complementa.

NoSQL (Not Only SQL): Tipo de base de datos no relacional.

RGPD (Reglamento General de Protección de Datos): Ley europea que protege los datos personales de los usuarios.

SSL (Secure Sockets Layer): Tecnología que cifra el contenido enviado por la red para que nadie pueda entenderlo. Usado con HTTPS

SSR (Server-Side Rendering): Técnica para generar páginas web desde el servidor y así poder ofrecer un servicio más personalizado.

UUID (Universally Unique Identifier): Identificador único que usa la aplicación a la hora de generar enlaces de partida.

VPC (Virtual Private Cloud): Red privada dentro de AWS que aísla y organiza los recursos del servidor.

Gracias.