GRADO EN INFORMÁTICA DE GESTIÓN Y SISTEMAS DE LA INFORMACIÓN

TRABAJO FIN DE GRADO

Universidad del País Vasco Euskal Herriko Unibertsitatea

VITORIA-GASTEIZKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE VITORIA-GASTEIZ

RAMSÉS.

Sistema de importación/exportación BD-JFLAP, gestión de idiomas y copias de seguridad

ALUMNO/A: Maider Pozo Yubero

DIRECTOR: Ismael Etxeberria Agiriano

CURSO: 2021-2022

FECHA: 12-09-2022

Resumen

En este documento se muestra cómo se ha ido desarrollando el Trabajo de Fin de Grado de la autora. El proyecto consiste en añadir funcionalidades y especificaciones a un área de RAMSÉS, aplicación web de simulación de máquinas abstractas. Es una más completa herramienta de uso académico para la asignatura de segundo curso Lenguajes, Computación y Sistemas Inteligentes, cubriendo la funcionalidad de una herramienta ya existente, JFLAP.

En el caso de este proyecto en cuestión se ha desarrollado la opción de importar cualquier fichero *JFLAP* o *JSON* a la base de datos. De igual manera se ha desarrollado la opción de exportar cualquier autómata a *JFLAP* o *JSON*.

Asimismo, se han añadido funcionalidades para etiquetar máquinas abstractas en la base de datos permitiendo asociarles información académica, como pueden ser colecciones de ejercicios o preguntas de exámenes. También se ha desarrollado un mecanismo adecuado para que RAMSÉS sea una herramienta multi-idioma.

Por último, se ha desarrollado un procedimiento para la realización de copias de seguridad de RAMSÉS para proteger las actividades académicas a pérdidas de información asociadas a incidentes de seguridad.

Cabe destacar que este proyecto está realizado en gran parte en el lenguaje de programación JavaScript tanto en la parte cliente como en la de servidor, apoyado en tecnologías como Node.js, Git, *JSON, SVG, CSS3 y HTML5*, junto con *MySQL* para el almacenamiento y recuperación de datos.

Por otro lado, se ha realizado la puesta en producción de la aplicación mediante máquinas virtuales en un servidor, mediante *Docker* y *Proxmox*. En estos momentos RAMSÉS está en producción en el servidor del departamento de LSI siendo accesible desde la Web.

Palabras clave: Autómatas; Intérprete; JavaScript; JFLAP; JSON; Software; RAMSÉS; Servidor; Docker; Proxmox; Turing

Índice de contenidos

Índice de Ilustraciones

Índice de tablas

Capítulo 1. Introducción

1.1 Título del proyecto

"RAMSÉS: Sistema de importación/exportación BD-JFLAP, gestión de idiomas y copias de seguridad".

RAMSÉS es el acrónimo de *"Recursive Abstract Machines Simulation EnvironmentS"*[30] .

1.2 Contextualización

1.2.1 Descripción del proyecto

Este proyecto ha sido llevado a cabo para realizar las tareas y exámenes de la asignatura de Lenguajes, Computación y Sistemas Inteligentes (LCSI) de una manera más automatizada. Esta asignatura es impartida en el Grado de Ingeniería Informática de Gestión y Sistemas de la Información, en la Escuela de Ingeniería de Vitoria Gasteiz.

En dicha asignatura se viene utilizando una herramienta llamada JFLAP[1]. Esta herramienta permite realizar la mayoría de ejercicios prácticos de la asignatura con expresiones regulares, autómatas y gramáticas, entre otros. Facilita el análisis práctico de los conceptos abstractos abordados, asimilándose con más facilidad.

El objetivo principal de la nueva plataforma de práctica es poder extender algunas de las funcionalidades de JFLAP, sobre todo para corregir automáticamente los ejercicios y para que sirva de plataforma de evaluación.

1.2.2 Antecedentes del proyecto

La idea surgió de la recopilación de posibles mejoras y recogida de errores del actual software principal utilizado en la asignatura de LCSI (Lenguajes, Computación y Sistemas Inteligentes) impartida en segundo curso por el tutor del presente proyecto, Ismael Etxeberria Agiriano. Dicho software es JFLAP, un programa precompilado en Java que permite diseñar y ejecutar máquinas abstractas: autómatas finitos, autómatas con pila y máquinas de Turing. Todo ello se realiza de forma gráfica mediante un editor de diagramas. En las ilustraciones, 1 y 2, se proporciona una muestra de los servicios que ofrece el software JFLAP.

[1] JFLAP acrónimo de *Java Formal Languages and Automata Package*

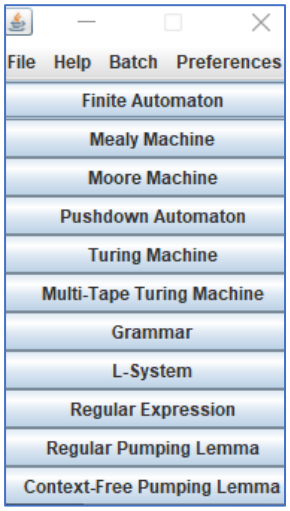

***Ilustración 1**: Menú JFLAP*

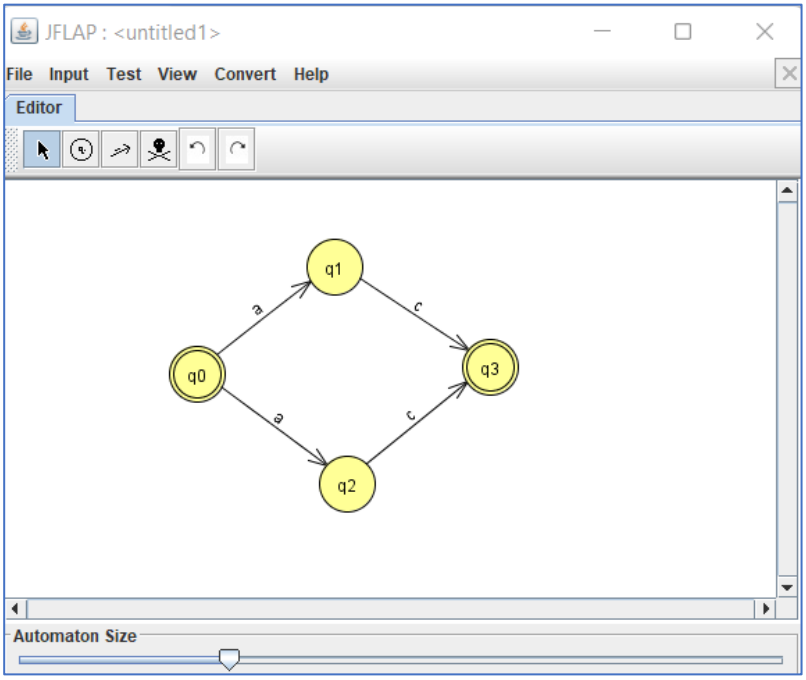

***Ilustración 2**: Diseño de autómata en JFLAP*

Tras años de impartición de la asignatura con la ayuda de dicha herramienta, el profesor consideró que el desarrollo de un entorno nuevo apoyado por un SGBD (Sistema de Gestión de Bases de Datos) podría mejorar el uso académico de la herramienta, proponiendo mantener las funcionalidades existentes, mejorar algunas de ellas y facilitar la evaluación del desempeño del alumnado. Por ello, la antigua alumna Estíbaliz Rodríguez de Yurre empezó con el desarrollo de la aplicación pertinente, y un año más tarde se ha seguido con su desarrollo [30].

1.2.3 Contexto y objetivos particulares

El proyecto RAMSÉS consiste en la realización de una aplicación web como entorno de desarrollo y simulación de máquinas abstractas.

Esta herramienta permite generar diagramas como representación visual de máquinas de diferente tipo: autómatas finitos deterministas, autómatas finitos no deterministas, máquinas de Turing y autómatas con pila.

RAMSÉS permite, a partir de una cadena de entrada o cinta de caracteres introducida por el usuario, saber si el autómata representado en el diagrama reconoce o no reconoce dicha cadena. Esto dependerá de variables como el tipo de máquina o el alfabeto reconocido por el autómata.

Para este TFG se propone desarrollar un mecanismo para importar cualquier fichero JFLAP o JSON a la base de datos. También se pretende permitir exportar cualquier representación de autómatas de RAMSÉS a *JFLAP (XML) o JSON*[2].

Asimismo, se requiere agrupar y etiquetar lenguajes formales para permitir asociarles información académica, como pueden ser colecciones de ejercicios o preguntas de exámenes. Esto llevará a cabo enriqueciendo el diseño de la base de datos y añadiendo funcionalidades que permitan gestionarlo. También se desarrollará un mecanismo adecuado para que se pueda hacer uso de RAMSÉS en distintos idiomas.

1.2.4 Objetivos y alcance del proyecto

Los objetivos son los siguientes:

- Conversión entre *JSON*, *JFLAP*, BD: Se trata de desarrollar mecanismos para cargar cualquier fichero *JFLAP* o *JSON* a la base de datos y de poder exportar cualquier lenguaje formal a *JFLAP* (si lo soporta) o *JSON*. Si es justificado, es posible que en las transformaciones se pierda información o no sea del todo equivalente (por ejemplo, la posición gráfica de los elementos o la información asociada a los ficheros) [14]. La información asociada no soportada por *JFLAP* puede ir en ficheros adicionales que permitan su tratamiento automático.
- Información asociada a los lenguajes: Se podrán agrupar y etiquetar lenguajes formales para permitir asociarles información académica, como pueden ser ejemplos, colecciones de ejercicios o preguntas de exámenes. Esto será un campo añadido a la base de datos.
- Cada lenguaje formal tendrá asociada la autoría del mismo. Por lo tanto, para guardar un lenguaje será necesario estar registrado en el sistema.

[2] JSON acrónimo de *JavaScript Object Notation*

- Gestión de idiomas: RAMSÉS debe integrar el potencial de utilizar cualquier idioma que se defina como soportado. Se han de desarrollar mecanismos e interfaces adecuados para que añadir un idioma nuevo sea solamente una cuestión de añadir la traducción de los textos a partir de las etiquetas existentes en otros idiomas.
- Monitorización de actividad: El sistema debe ser capaz de monitorizar la actividad de los usuarios, es decir, permitir estimar cuánto tiempo ha dedicado cada usuario a las actividades.
- Copias de seguridad: La utilización de RAMSÉS para actividades académicas implica el aseguramiento de la preservación de la información frente a incidentes de seguridad. Para ello RAMSÉS dispondrá de mecanismos para enviar la información a la correspondiente base de datos cuando esté lista, además de realizar copias periódicas y automáticas.
- Para realizar copias de seguridad se dispondrá, por un lado, de la distribución exacta instalada en un lugar (ficheros de la aplicación y estructura de datos), junto a los detalles de configuración del entorno de trabajo (*Docker, Proxmox*, ...). No será preciso realizar copias de seguridad de estos apartados mientras no se modifiquen. Por otro lado, la modificación de estas partes implicará la actualización de la copia correspondiente.

1.2.5 Entorno de desarrollo del proyecto

El sistema RAMSÉS debe ser desplegado en *Node.js* [22] , en un servidor local en el puerto deseado, el cual será migrado a uno de los servidores de la universidad proporcionado por el tutor del presente trabajo de fin de grado.

Una copia del código debe ser registrada en un repositorio remoto de *GitHub*, al que tendrá acceso la autora y el tutor del proyecto, y las personas autorizadas por este último a partir de la entrega del proyecto.

En cuanto a medios físicos, se proporcionará a la Escuela una copia física, impresa y encuadernada del documento, y un CD con la memoria del proyecto.

1.2.6 Organización de la memoria

El documento actual está organizado en siete capítulos que se describen brevemente a continuación.

Capítulo 1 - Introducción: En este capítulo se realiza la descripción y contextualización del proyecto.

Capítulo 2 – Viabilidad del proyecto: Se realiza la planificación del proyecto, la estimación de costes y el análisis de riesgos.

Capítulo 3 – Desarrollo técnico: En este capítulo se detallan tanto las tecnologías como las herramientas utilizadas en el proyecto.

Capítulo 4 – Análisis técnico: Consiste en la explicación del *back end* y la puesta en producción.

Capítulo 5 – Trabajo realizado: En este capítulo se detalla el trabajo realizado por la autora.

Capítulo 6 – Análisis final: Se realiza la conclusión y un pequeño estudio sobre las líneas futuras.

Capítulo 2. Viabilidad del proyecto

2.1 Planificación del proyecto

En este capítulo se describe la estructura de descomposición del trabajo utilizada en la realización de la aplicación; las fases, tareas y entregables del proyecto y cuál será la duración estimada de cada una de estas, así como la agenda de trabajo y los recursos materiales empleados.

2.1.1. Análisis del proyecto

Al comienzo del proyecto se decidieron cuáles eran las funcionalidades a llevar a cabo para continuar con RAMSÉS. También se establecieron reuniones periódicas con el director del proyecto para subsanar dudas y si era preciso, modificar las especificaciones.

En este curso académico, el alumno Gaizka Valle y la autora, Maider Pozo, han seguido con la implementación de RAMSÉS a partir de la implementación existente. Las tareas se pueden observar desglosadas en la siguiente tabla 1.

Tabla 1: *Tabla de tareas*

Tarea	Dependencia	
0	Proyecto RAMSÉS	
1	Análisis del proyecto	
1.1	Estudio de requerimientos	
1.2	Reparto de funcionalidades	1.1
1.3	Declaración de alcance y objetivos	1.2
2	Diseño	1.3
2.1	Diseño de requisitos	1.2
2.2	Análisis del proyecto anterior	1.2
2.3	Análisis de software y requerimientos	2.2
3	Implementación	2.3
3.1	Puesta en producción	2.2
3.2	Mantenimiento de base de datos	2.3
3.3	Conversión entre XML y JSON	2.2
	Estudio de información asociada al lenguaje	2.2
3.4	Gestión de idiomas	2.2
3.5	Importación de autómatas	2.3
4	Etapa de pruebas con cliente	3.3
4.1	Reunión con cliente	3
4.2	Implementación de cambios	4.1
5	Memoria	4.2

Al contar con dos desarrolladores en el proyecto, las especificaciones fueron divididas en dos bloques. En este trabajo de fin de grado del proyecto correspondiente a este TFG se detalla el bloque del desarrollo *back end*.

2.1.2. Calendario del proyecto

En cuanto al calendario del proyecto, se hará una estimación de cuales serás los días que se considerarán laborables en el pertinente proyecto. Teniendo en cuenta que el proyecto comenzó el 13/10/2022, los días no laborables se reducen a los siguientes.

- 1 de noviembre: Día de Todos los Santos
- 6 de diciembre: Día de la Constitución Española
- 8 de diciembre: Inmaculada Concepción
- 24 de diciembre: Navidad
- 1 de enero: Año Nuevo
- 6 de enero: Epifanía del Señor
- 14 de abril: Jueves Santo
- 15 de abril: Viernes Santo
- 18 de abril: Lunes de Pascua
- 28 de abril: San Prudencio
- 25 de julio: Santiago Apóstol
- 5 de agosto: Virgen Blanca
- 15 de agosto: Festividad de la Asunción de la Virgen

El calendario del proyecto completo teniendo en cuenta los días laborables nombrados quedaría de la manera en la que se puede observar en la ilustración 3.

2.1.3. Diagrama de Gantt

Tomando en cuenta el análisis realizado en el apartado anterior se ha generado un diagrama de Gantt de las Ilustraciones 3-7 con el uso de la herramienta *ProjectLibre* [1] .

	Nombre	Duracion	Inicio	Terminado	Predecesores
1	**Proyecto RAMSÉS**	223 days?	**13/10/21 8:00**	**9/09/22 17:00**	
2	**Análisis del proyecto**	9,125 days?	**13/10/21 8:00**	**26/10/21 9:00**	
3	Estudio de requerimientos	6 days?	13/10/21 8:00	20/10/21 17:00	
4	Reparto de funcionalidades	1 day?	22/10/21 8:00	22/10/21 17:00	3
5	Declaración de alcance y objetivos	1 day?	25/10/21 9:00	26/10/21 9:00	4
6	**Diseño**	33,75 days?	**26/10/21 10:00**	**14/12/21 17:00**	
7	Diseño de requisitos	33 days?	26/10/21 10:00	14/12/21 10:00	5
8	Análisis del proyecto anterior	10,875 days?	27/10/21 9:00	11/11/21 17:00	5
9	Análisis de software y requerimientos	22 days?	12/11/21 8:00	14/12/21 17:00	8
10	**Implementación**	170 days?	**12/12/21 8:00**	**24/08/22 17:00**	
11	Puesta en producción	26,875 days?	15/12/21 8:00	31/01/22 16:00	9
12	Mantenimiento de base de datos	35 days?	12/12/21 8:00	8/02/22 17:00	8
13	Conversión entre XML y JSON	44 days?	15/12/21 8:00	23/02/22 17:00	9
14	Estudio de información asociada al lenguaje	74 days?	13/03/22 8:00	28/06/22 17:00	9
15	Gestión de idiomas	11 days?	15/07/22 8:00	1/08/22 17:00	9
16	Importación de autómatas	20 days?	25/07/22 8:00	24/08/22 17:00	8
17	**Etapa de pruebas con el cliente**	12 days?	**25/08/22 8:00**	**9/09/22 17:00**	
18	Reunión con el cliente	8 days?	25/08/22 8:00	5/09/22 17:00	10
19	Implementación cambios sugeridos	4 days?	6/09/22 8:00	9/09/22 17:00	18
20	Memoria	60 days?	15/06/22 8:00	9/09/22 17:00	

***Ilustración 3**: Lista de tareas y sus dependencias*

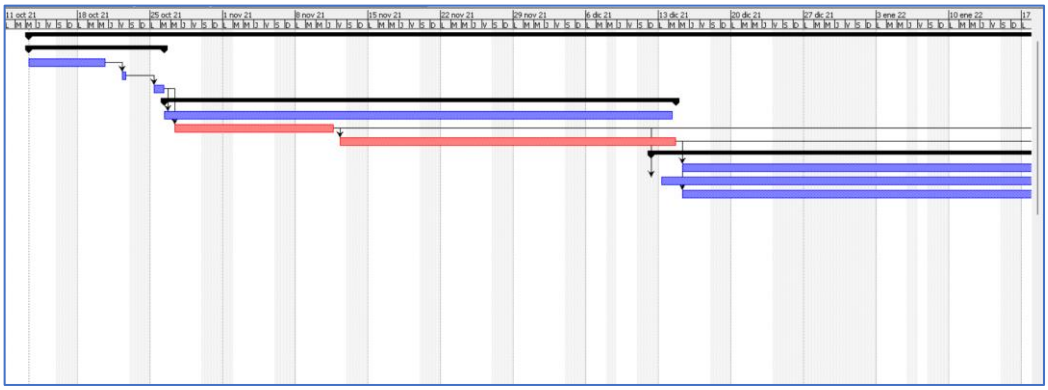

***Ilustración 4**: Diagrama de Gantt en ProjectLibre (I)*

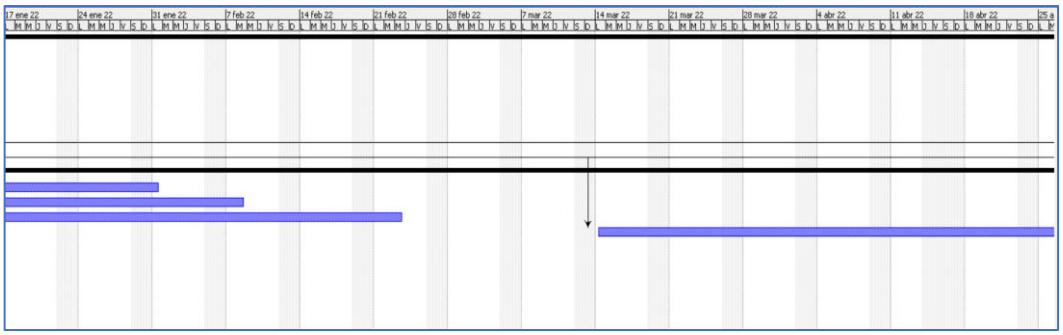

***Ilustración 5:** Diagrama de Gantt en ProjectLibre (II)*

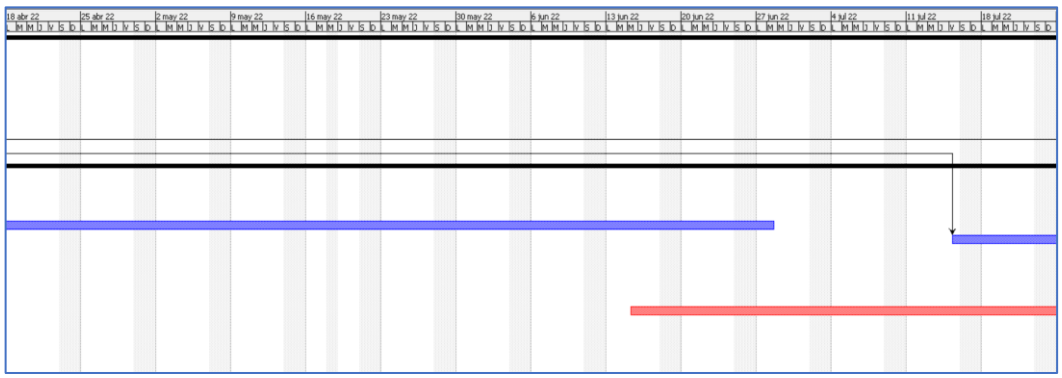

***Ilustración 6:** Diagrama de Gantt en ProjectLibre (III)*

Ilustración 7: *Diagrama de Gantt en ProjectLibre (IV)*

2.1.4. Tareas del proyecto

En esta sección se detallarán las tareas a realizar y sus responsables. El proyecto cuenta con tres responsables:

- Maider Pozo: Programador junior
- Gaizka Valle: Programador junior
- Ismael Etxeberria: Director del proyecto

Las tareas encomendadas a cada responsable están especificadas en la Tabla 2.

Tabla 2: *Responsables de tareas*

Tarea a realizar	Responsable
Definición de especificaciones y requisitos	Ismael Etxeberria
Prueba de nuevas funcionalidades	Ismael Etxeberria
Puesta en producción (entorno)	Maider Pozo y Gaizka Valle
Adaptación de la lógica	Gaizka Valle
Implementación funcionalidades	Gaizka Valle
Puesta en producción (base de datos)	Maider Pozo
Mantenimiento base de datos	Maider Pozo
Conversión entre XML y JSON	Maider Pozo
Gestión de idiomas	Maider Pozo
Importación de autómatas en XML	Maider Pozo
Estudio almacenamiento de datos e información en la base de datos	Maider Pozo

2.2 Estimación de costes

2.2.1 Recursos humanos

Para determinar los costes de recursos humanos en el proyecto, se ha utilizado como referencia el convenio colectivo de la empresa Izmar SLU del Boletín Oficial del estado publicado el 17 de setiembre de 2021 [3] .

Según la **disposición 15125** de la Dirección General de Trabajo, la página 28 nos presenta la siguiente tabla salarial para el Grupo Profesional 3 y salario funcional /nivel 2. Esto se puede observar en la ilustración 8, donde se muestran los salarios según el nivel del trabajador.

Grupo profesional 3. Técnicos A

Función	Salario grupo	Salario función/ nivel 1	Salario función/ nivel 2	Salario función/ nivel 3
Administrador de Sistemas.	16.321,60	1.530,15	5.100,50	8.160,80
Técnico preventa.	16.321,60	2.040,20	7.140,70	10.201,00
Técnico gestor de proyectos.	16.321,60	2.040,20	5.100,50	9.180,90
Técnico especialista en gestión.	16.321,60	2.040,20	5.100,50	9.180,90

***Ilustración 8**: Salarios Boletín Oficial del estado publicado el 17 de setiembre de 2021*

$$16.321{,}60 \; + \; 5.100{,}50 \; = \; 21.422{,}10 \; €/año$$

Esto es lo que se conoce como bruto anual del trabajador. En el supuesto de que el/la programador/a esté contratado/a, el abono a la seguridad social eleva el coste a:

$$21.422{,}10 \; x \; 1{,}30 = \; 27.848{,}73€/\text{año}$$

Según el Boletín oficial del estado una jornada anual consta de 1750 horas[3] $(27.848{,}73 \; €/ \; 1.750 \; horas)$. Teniendo en cuenta esto último el salario/hora asciende a **15,91 €/Hora.**

Los costes de la mano de obra serían los siguientes:

- Programador junior: 15,91 €/horas

Según el BOE consultado la dedicación anual es de 1.750 horas. Con esto el coste por hora de trabajo para una programadora junior sería:

$$\frac{27.848.73 \; €/año}{1.750 \; horas/año} \; = \; 15{,}91 \; €/hora$$

[3] BOE establece que la cantidad de horas laborales son 1.750 horas

Por lo tanto, el coste total de recursos humanos sería el siguiente.

$$585\ h\ \times 15.91\ \frac{€}{hora} = 9.307{,}35\ €$$

2.2.2 Recursos materiales

En esta sección se definirán las amortizaciones de los costes de recursos materiales, tanto costes de software como de hardware. Los costes de amortización se tendrán en cuenta a 2 años de horas trabajadas.

Según el Boletín Oficial del Estado, BOE, la cantidad de horas laborales son 1.750 horas. En este caso al tener 585 horas trabajadas, la amortización a tres años serán 1.755 horas.

Para el cálculo del coste unitario de amortización y la amortización total se utilizarán las siguientes fórmulas matemáticas. Hacemos una estimación del equipo de tres años. 1.750h en tres años equivaldrían a 5.250 horas.

$$\text{Coste Unitario de Amortización} = \text{Coste unitario} / \text{Horas de Amortización}$$

$$\text{Amortización total} = \text{horas de uso (117 d} \times 5\text{h} \sim 585\text{h)} \times \text{Amortización unitaria} \times \text{Cantidad}$$

Todo el software utilizado para el desarrollo técnico del proyecto es de uso gratuito por lo cual no se nombrará ningún tipo de software en estos costes. En la Tabla 4 se puede observar cual sería el coste material en este proyecto.

***Tabla 3**: Coste recursos materiales*

Elemento	Coste unitario	Cantidad	Años	Amortización Ud.	Amort. total
PC ACER aspire	599,00 €	1	3	$1.750\ x\ 3\ =\ 5.250\ h$ $\frac{599}{5.250} = 0{,}11\ €/h$	$585\ h$ $\times 0{,}11\ \frac{€}{h} x\ 1PC$
				TOTAL	64,35 €

Para el cálculo de los costes de soporte de internet según los cálculos realizados, la tarifa promedio de internet con la compañía Pepephone es de 29 €/mes para 600 Mb de Fibra. Se ha realizado el coste de soporte de internet por hora, por lo que serían 0,26 €/hora[4]. El cálculo se hace teniendo en cuenta las 585h trabajadas. En la Tabla 5 se muestra el coste del soporte de internet en el proyecto.

[4] El cálculo es: $29\frac{€}{mes} \div 22\ días = 1{,}31\frac{€}{día} \div 5\frac{h}{día} = 0{,}26\ €/h$

Tabla 4: Coste soporte de internet

Elemento	Coste unitario	Horas	Total
Soporte de internet	$0{,}26\ €/hora$	$585h$	146,01 €

Para el coste del consumo eléctrico se ha realizado de forma similar. El consumo del equipo más una posible lámpara en determinados momentos estimamos un consumo de 200 watios en las horas de trabajo teniendo un consumo de energía de 0.2 Kwh.

El precio de la energía eléctrica durante este año 2022 ha tenido muchas variaciones en el precio, pero estimamos un precio medio incluido el IVA de 15 céntimos de euro el Kwh. En la tabla se obtiene el coste de la energía eléctrica en el proyecto.

Tabla 5: Coste energía eléctrica

Elemento	Horas	KWH	Coste unitario	Total
Consumo de electricidad	585	$585\ h\ x\ 0{,}2 = 117\ Kwh$	$0{,}15€\ /\ Kwh$	17,55 €
			TOTAL	17,55 €

En la Tabla 7 se puede observar el coste general tanto de los recursos humanos como los materiales.

Tabla 6: *Coste general recursos*

Recursos		Coste
Humanos		9.307,35 €
Materiales	**Equipo informático**	64,35 €
	Soporte de internet	146,01 €
	Energía eléctrica	17,55 €
	Total	9.535,26 €

2.2.3 Presupuesto

Teniendo en cuenta los costes de recursos humanos y materiales calculados anteriormente, se calculará el presupuesto del proyecto. Para ello se tiene en cuenta un 21 % de IVA y unos beneficios de 15%. En la Tabla 8 se muestra el presupuesto completo.

Tabla 7: Presupuesto

Concepto	Importe
Recursos humanos	9.307,35 €
Recursos materiales	227,91 €
Suma	9.535,26 €
Beneficios (15%)	1.430,28 €
Subtotal	10.965,54 €
IVA (21 %)	2.302,76 €
Total	13.268,30 €

2.3 Análisis de riesgos

La gestión de los riesgos es una parte integral de la dirección del proyecto, siendo un elemento clave en el proceso de toma de decisiones. El análisis de riesgos es el estudio de las amenazas, los eventos, los daños y las posibles consecuencias que puede llegar a tener poner en marcha un proyecto determinado. Es importante especificar que se entiende por amenaza, agente externo o vulnerabilidad que pudiera alterar el proceso o afectar a uno de los elementos del proyecto [26] .

Antes de iniciar la planificación del proyecto, se han analizado los riesgos potenciales para el desarrollo del proyecto. De esta forma, se puede crear un plan de contingencia o, en su defecto, tomar medidas cautelares en caso de que finalmente se produzcan estos percances.

Para ello, se analizan y evalúan los riesgos teniendo en cuenta su probabilidad y gravedad. Por lo tanto, la probabilidad y la gravedad se evalúan mediante uno de los siguientes valores: bajo, medio o alto. El resultado será un valor entre 1 y 5. La Tabla 9 muestra la tabla de equivalencias de los riesgos.

Tabla 8: Tabla de equivalencias

Valor	Equivalencia
1	**Riesgo trivial:** No hay necesidad de tomar acción.
2	**Riesgo tolerable**: No es grave, pero es preferible reducir la probabilidad de que ocurra.
3	**Riesgo moderado**: Se deben tomar medidas preventivas para reducir la probabilidad.
4	**Riesgo importante:** Debe ser prioritario tomar medidas preventivas para evitarlo.
5	**Riesgo grave:** No se puede continuar o empezar ningún tipo de acción sin antes haber solucionado el riesgo.

Al tener en cuenta los criterios explicados anteriormente se procede a realizar una lista de los posibles riesgos a los que el proyecto está expuesto y las medidas que sería conveniente tomar. En la Tabla 10 se muestran los riesgos del proyecto, junto con su valoración y medidas de prevención.

Tabla 9: *Tabla de riesgos*

Riesgo	Gravedad	Probabilidad	Valoración	Medida de prevención
Rotura del pc de trabajo del desarrollador	Media	Media	2	Al tener un único pc, cada semana se realizará una copia de todo el material necesario y actualizado en el pc.
Baja por enfermedad del programador/a	Baja	Baja	1	Se tomarán las pertinentes medidas acordadas en todo momento por las autoridades sanitarias, así cómo aplicar el teletrabajo en la medida de lo posible. [5]
Problemas en el desarrollo de una tarea	Media	Media	3	Se realizarán reuniones cada 2 semanas con el director del TFG para intentar subsanar todas las dudas que puedan surgir en la implementación.
Problemas por baja del cliente	Baja	Baja	2	Al iniciar el proyecto y después de cada reunión se le pedirá al cliente un documento con las especificaciones requeridas y con vistas a futuro para así tener material en caso de tener problemas para asistir a reuniones.
Pérdida de datos e información	Alta	Baja	3	Realización de copias de seguridad programadas con frecuencia frecuente y en varios dispositivos diferentes.
Tareas no programadas	Baja	Alta	3	Se asignará un día a la semana para salirse de la especificación inicial e intentar subsanar dichas tareas.
Aumento de los requisitos	Media	Alta	4	A lo largo de todo el desarrollo se modularizarán todos los requisitos

[5] Al vivir en una sociedad con grandes riesgos por las actuales emergencias sanitarias como la Covid-19 (SARS-CoV-2)

Las vulnerabilidades detectadas explicadas extensamente son las siguientes.

2.3.1 Perdida de datos

- **Descripción**: Fallo en el equipo de trabajo del desarrollador que suponga la pérdida de todos los datos del proyecto: código fuente, documentación.
- **Alcance**: Muy grave. Esta situación supondría comenzar de nuevo el proyecto desde cero, incluyendo la instalación de las herramientas necesarias, con el retraso que conllevaría.
- **Medidas preventivas**: Almacenar copias de seguridad de la documentación fuera del sistema de archivos del equipo, mantener el código fuente de la aplicación en un repositorio remoto y siempre actualizado.
- **Medidas correctoras**: Intentar recuperar la información de los medios de almacenamiento.

2.3.2 Tareas no programadas

- **Descripción**: Aparición de nuevas tareas o requisitos importantes durante el desarrollo del proyecto, o modificación severa de los ya existentes.
- **Alcance**: Grave o perjudicial, en función de la complejidad de los nuevos requisitos.
- **Medidas preventivas:** Realizar una correcta recogida de requisitos tanto funcionales como no funcionales previa al comienzo del proyecto, y una buena gestión de proyecto. Supondría afectar los costes y los plazos de entrega.
- **Medidas correctoras**: Priorizar tareas, aumentar los recursos disponibles si fuese posible.

2.3.3 Problemas de codificación

- **Descripción**: Problemas relativos al código fuente (funcionalidad que se solapa, por ejemplo) que puede llegar a paralizar el proyecto.
- **Alcance:** Perjudicial, podría suponer un retraso en el proyecto.
- **Medidas preventivas:** Realizar un buen estudio previo de las tecnologías y herramientas de trabajo. Intentar evitar dependencias entre partes de la aplicación o funcionalidades, para poder continuar trabajando en otra hasta que se supere el bloqueo.
- **Medidas correctoras:** Compensar el tiempo perdido con horas extras posteriores, si se quieren mantener los plazos de entrega.

2.3.4 Enfermedad del desarrollador

- **Descripción**: Baja por enfermedad o accidente del desarrollador, inhabilitándolo de forma temporal.
- **Alcance**: Perjudicial. Puede ocasionar un retraso en el proyecto si la afección le impide trabajar por completo.
- **Medidas preventivas**: No previsible.
- **Medidas correctoras:** Compensar el tiempo de baja con horas extras posteriores, si se quieren mantener los plazos de entrega.

Capítulo 3. Desarrollo técnico

En esta sección se estudian las herramientas y tecnologías utilizadas para el desarrollo de RAMSÉS.

3.1 Tecnologías

3.1.1 HTML/HTML5

HTML (*HyperText Markup Language*) es uno de los lenguajes de marcado más utilizados para la creación de sitios web, definiendo su estructura mediante etiquetas. En HTML/HTML5 los elementos que componen una página web pueden ser desde texto plano, a vídeos u otros contenidos multimedia, pasando por imágenes, enlaces, botones o formularios [13] .

Los elementos constan de dos propiedades básicas: atributos y contenido. Cada atributo y contenido tiene ciertas restricciones, en algunos casos pudiendo tomar un único valor de entre algunos predefinidos. Las etiquetas antes mencionadas sirven para delimitar el inicio y el fin de un elemento. Aunque hay excepciones, cada elemento tiene generalmente una etiqueta de apertura (<*mi-elemento*>) y una otra de cierre (</*mi-elemento*>).

Los atributos del elemento están contenidos en la etiqueta de inicio y el contenido está ubicado entre las dos etiquetas (<*mi-elemento atributo*="valor">Contenido</*mi-elemento*>)

Los documentos HTML normalmente son archivos con la extensión .html o .htm, que se pueden visualizar en cualquier navegador web. El navegador lee el archivo .html, lo interpreta y lo mostrará al usuario. La Ilustración 9 muestra el logo de la tecnología HTML.

Ilustración 9: *HTML logo*

3.1.2 CSS/CSS3

CSS, acrónimo de *Cascading Style Sheets*, es un lenguaje que sirve para definir la apariencia de un documento *HTML*, *XML* o derivados de este último [18] .

La sintaxis se compone de un selector (es el elemento al que se aplican los estilos) seguido de la declaración de estilos (características de diseño a añadir o modificar).

Al ser una página web con únicamente HTML muy básica, es necesario CSS para poder aplicar estilos, como el fondo, los colores, el tamaño de letra y las animaciones, y *JavaScript* para definir la funcionalidad de los elementos de la estructura.

Es utilizado para diseñar y dar estilo a las páginas web: nos permite alterar características de la fuente como su tamaño o color, el fondo de la página, tamaño de los elementos, incluir animaciones y otras características decorativas. La Ilustración 10 muestra el logo de la tecnología css.

***Ilustración 10**: Logo css*

3.1.3 Media queries

Parte del potencial de *CSS3* reside en la creación de sitios responsivos, es decir, que la aplicación y sus páginas puedan visualizarse correctamente en cualquier dispositivo.

Para ello existen las *medias queries*, que son reglas específicas para modificar la página o aplicación web en función del tipo de dispositivo (portátil, tableta, smartphone, monitor) o de características y parámetros específicos, (como la resolución de la pantalla, el ancho del *viewport* del navegador, o la densidad de píxeles de la pantalla) [19] .

3.1.4 Flexbox

Flexbox es un modelo de diseño *CSS3* que se caracteriza por permitir que el alto y ancho de los elementos de la página se ajuste lo mejor posible al espacio disponible, de una forma mucho más sencilla que la tradicional [19] .

Se define un contenedor (denominado contenedor *Flex*) desde el cual vamos a poder especificar la alineación, el tamaño y la dirección de los elementos que hay dentro, así como distribuir el espacio restante que hay entre ellos.

3.1.5 Web Components

Son un conjunto de tecnologías para desarrollar interfaces y funcionalidades en el lado del cliente utilizando para ello las tecnologías nativas que soporta cualquier navegador. Estas son: *HTML, CSS y JavaScript*.

Estas tecnologías permiten crear componentes reutilizables, y estos permiten definir el comportamiento de los elementos, y más tarde personalizarlos desde *JavaScript*. Estos elementos funcionan de forma independiente al resto del código [16] .

El renderizado de cada elemento se realiza aparte del árbol DOM principal, para evitar posibles conflictos de estilo y funcionalidad con otras partes de la aplicación. De esta forma, a cada elemento se le pueden añadir estilos, oyentes de eventos, u otros elementos hijos de forma encapsulada y privada para el resto de la aplicación. Permite la creación de contenido dinámico, evita código repetido y aligera la aplicación final. La Ilustración 11 muestra el logo de la tecnología *Web Components.*

Ilustración 11: *Logo Web Components*

3.1.6 JSON

Es el acrónimo de *JavaScript Object Notation.* Es un formato de archivo de extensión **json.**, está basada en texto plano para representar estructuras de datos en la sintaxis de objetos de *JavaScript* – aunque es independiente de este y de cualquier lenguaje de programación.

Es un formato fácil de leer y entender, ligero (en bytes) y que soporta seis tipos diferentes de datos: nulo, números, cadenas, booleanos, matrices y objetos [15] .

La simplicidad de *JSON* ha dado lugar a la generalización de su uso, especialmente como alternativa a *XML*. La Ilustración 12 muestra el logo de la tecnología *JSON*.

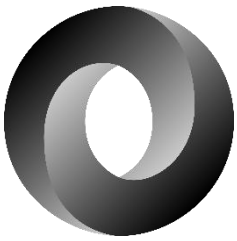

Ilustración 12: *Logo JSON*

3.1.7 SVG

Formato de gráficos vectoriales escalables (*Scalable Vector Graphics*), que constituye un estándar basado en *XML*. Se almacena en ficheros de texto plano y define una estructura de objetos (con una sintaxis de apariencia similar a la que tiene *HTML*) que puede modificarse manual y dinámicamente mediante código [29] . La Ilustración 13 muestra el logo de la tecnología *SVG*.

Ilustración 13: *Logo SVG*

3.1.8 Node.js

Node.js es un entorno multiplataforma de *JavaScript* (por ello la terminación .js) del lado del servidor, con una arquitectura orientada a eventos, y basado en el motor V8 de Google.

Node.js es capaz de gestionar múltiples peticiones y conexiones de forma eficaz: como alternativa a otras tecnologías del lado del servidor, que ejecutan cada petición en un hilo independiente, *Node.js* emplea un solo hilo y un bucle de eventos asíncrono, en el que las nuevas peticiones se tratan como eventos. La más conocida es la de permitir ejecutar JavaScript en el servidor. Las Ilustraciones 14 y 15 muestran el logo de la tecnología Node.js y un esquema del *server* de este.

***Ilustración 14**: Logo Node.js*

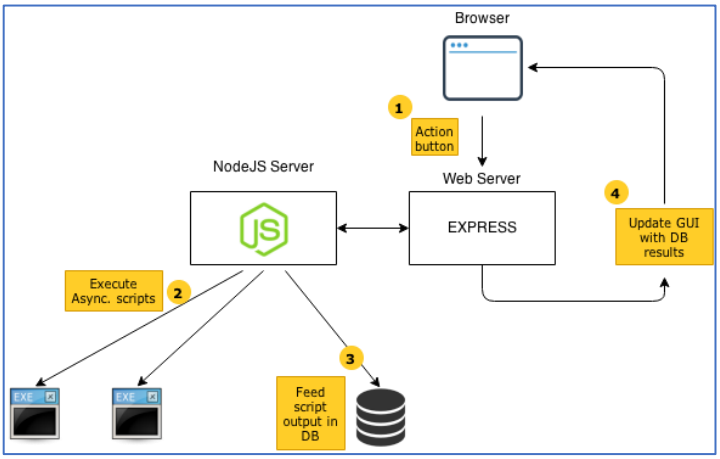

***Ilustración 15:** Esquema Server*

Otra de las características más importantes es su gestor de paquetes de terceros, *Node Package Manager (NPM)*. Este gestor otorga acceso a un conjunto de librerías muy extenso, gratuito y que son generadas a partir de la colaboración de los usuarios de su comunidad.

3.1.9 SQL

SQL responde a las siglas de *Structured Query Language*. Es un lenguaje de dominio utilizado en bases de datos relacionales [27] .

Sirve para acceder a información almacenada en las bases de datos, y también permite realizar operaciones de inserción, selección, borrado y actualización de datos. La Ilustración 16 muestra el logo de la tecnología SQL.

***Ilustración 16:** SQL*

3.1.10 GIT

Git es un software de control de versiones, de software libre. Esta tecnología está pensada para almacenar y gestionar nuestro código en un repositorio (local o remoto). Permite ramificar el desarrollo, y guardar código en una línea de progreso diferente a la principal, para aislar posibles errores o probar cambios. Dichas ramificaciones pueden fusionarse en cualquier momento a la línea de progreso principal, denominada *master* [12] . La Ilustración 17 muestra el logo de la tecnología GIT.

***Ilustración 17**: Git logo*

3.2 Herramientas

3.2.1 Visual Studio Code

Visual Studio Code, también conocido como *VS Code*, es un editor de código fuente creado por *Microsoft* para *Windows, Linux y macOS*. Es un editor de código desarrollado por *Microsoft*, multiplataforma, gratuito y de código abierto. Cabe señalar que no es un IDE puesto que no compila el código, es un editor [31] .

Admite la instalación de extensiones (en su mayoría creadas por miembros de la comunidad) que ayuden con el lenguaje de programación deseado, por ejemplo, recomendaciones de autocompletado, sintaxis de colores, tabulación automática, y un largo etcétera. Cuenta también con bastantes opciones de depuración de código, y de integración con *Git* (y otras plataformas de control de versiones). La Ilustración 18 muestra el logo de la herramienta *VSCode*.

***Ilustración 18:** VS Code logo*

3.2.2 MySQL Workbench

MySQL Workbench es un software gratuito creado por la empresa *Sun Microsystems*, esta herramienta permite interactuar de forma muy visual con bases de datos *MySQL*, sustituyendo la tradicional línea de comandos.

Su principal uso es el de modelado y visualización de datos: tablas, columnas, y consultas, a la par que gestionar scripts *SQL* (ejecutar y generar otros scripts a partir del modelo relacional creado) y administrar el servidor y las conexiones. La Ilustración 19 muestra el logo de la herramienta MySQL.

Ilustración 19: *MySQL Workbench logo*

3.2.3 GitHub

GitHub es un servicio de control de versiones y desarrollo de software colaborativo basado en *Git*. En este caso se ha utilizado el servicio de gestión y *host* de repositorio privado e integración continua, pero tiene muchos otros servicios cómo despliegue automático o automatización de procesos básicos. La Ilustración 20 muestra el logo de la herramienta *GitHub*.

Ilustración 20: *GitHub logo*

3.2.4 Proxmox

Proxmox Virtual Environment es un entorno de virtualización de servidores de código abierto. Es una distribución de *GNU/Linux* basada en *Debian*, con una versión modificada del *Kernel Ubuntu LTS 1* y permite el despliegue y la gestión de máquinas virtuales y contenedores. *Proxmox VE* incluye una consola Web y herramientas de línea de comandos, y proporciona una

API REST para herramientas de terceros [24] . La Ilustración 21 muestra el logo de la herramienta PROXMOX.

Ilustración 21: Proxmox logo

3.2.5 Docker

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.

Docker utiliza características de aislamiento de recursos del kernel Linux, tales como *cgroups* y espacios de nombres (**namespaces**) para permitir que "contenedores" independientes se ejecuten dentro de una sola instancia de *Linux*, evitando la sobrecarga de iniciar y mantener máquinas virtuales [7] . La Ilustración 22 muestra el logo de la herramienta docker.

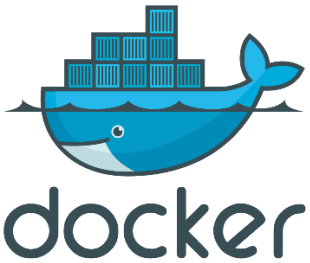

Ilustración 22: *Docker logo*

3.2.6 AnyConnect

AnyConnect Secure Mobility Client es un producto modular de software para terminales líder en la industria. No solo proporciona acceso a VPN a través de los protocolos *Secure Sockets Layer* (SSL) e *IPsec IKEv2*, sino que también ofrece seguridad mejorada a través de diversos módulos integrado [5]. La Ilustración 23 muestra el logo de la herramienta AnyConnect.

***Ilustración 23**: Anyconnect logo*

3.2.7 PuTTY

PuTTY es un emulador de terminal gratuito que admite varios protocolos de red tal como SSH. Esto te permite correr comandos *UNIX* en un servidor el cual no está disponible en una conexión mediante un cliente *FTP* [9]. La Ilustración 24 muestra el logo de la herramienta *PuTTY*.

***Ilustración 24:** PuTTY logo*

3.2.8 ProjectLibre

ProjectLibre es un software de administración de proyectos de código abierto, similar a *Microsoft Project*. ProjectLibre corre sobre la plataforma *Java*, lo que permite ejecutarlo en varios sistemas operativos. Esta herramienta se utiliza para los diagramas de Gantt [23] .La Ilustración 25 muestra el logo de dicha herramienta.

***Ilustración 25:** ProjectLibre logo*

3.2.9 Navegadores para pruebas

Actualmente RAMSÉS solo está operativo en *Google Chrome* y en *Safari*. Esto es debido a que existen dependencias instaladas que no son soportadas por otros navegadores. No es posible migrarlo a otros navegadores ya que perdería funcionalidades.

Ilustración 26: Google Chrome logo

3.3 Resumen de tecnologías

En la Tabla 11 se muestra un resumen de las tecnologías escogidas y el concepto al que responden.

Tabla 10: Resumen de tecnologías utilizadas

Concepto	Tecnología u Herramienta
Lenguaje programación principal	JavaScript
Editor de código	Visual Studio Code
Servidor	Express Node.js
Otros lenguajes utilizados	HTML, CSS
Tratamiento de datos	JSON, XML, SQL
Base de datos	MySQL
Control de versiones	GitHub
Gestión del proyecto	ProjectLibre
Sistema Operativo	Windows 10, Linux

Capítulo 4. Análisis técnico

4.1 Integración continua

La integración continua, también llamada *Continuous Integration*, es la metodología de automatizar la integración o actualización de código en un proyecto de software con varios colaboradores [2].

Para ello se ha utilizado la fusión (*Merge Request*) que proporciona GitHub para fusionar el código de una rama a la rama de desarrollo o principal.

El proceso es el siguiente: en el repositorio de código correspondiente se dispone de una rama principal a la cual se restringe subir código si no es a través de fusión, master. Por lo cual, cada integrante del grupo crea ramas a partir de la principal para desarrollar la funcionalidad específica y al subir la rama y una vez pasen los test unitarios solicita la fusión a otro integrante del proyecto. Entonces el integrante al que se le solicita la fusión revisa los cambios propuestos y acepta la fusión o comenta posibles modificaciones.

Finalmente, en caso de aceptar la fusión el código se fusiona a la rama principal y se borra la rama creada para el desarrollo y así se consigue que todo el mundo conozca todas las funcionalidades y la detección de posibles errores antes de pasar a producción el código.

4.2 Base de datos

En cuanto a la base de datos, cabe destacar que en el proyecto inicial el diseño inicial de la base de datos era el que se observa en la Ilustración 27.

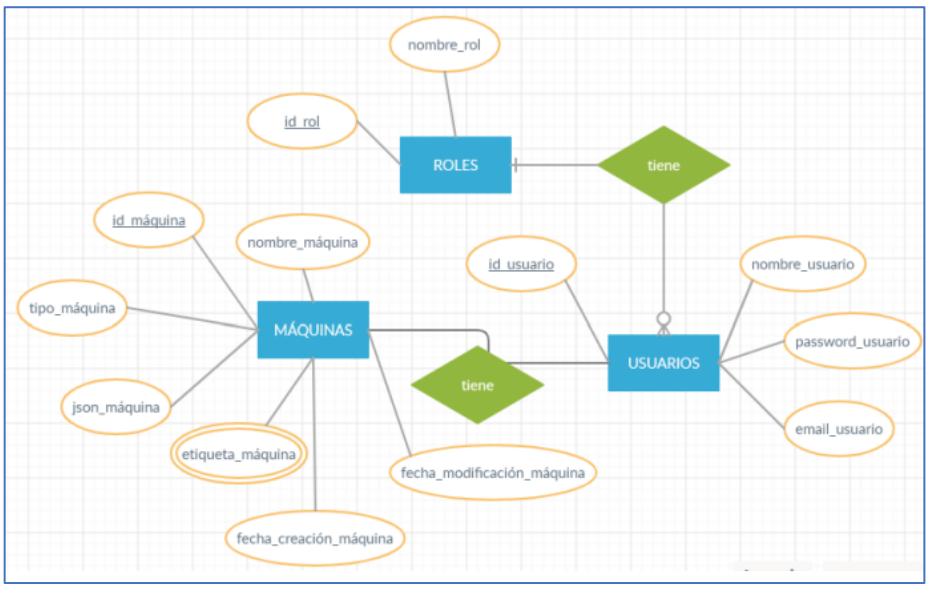

***Ilustración 27**: Modelo ER proyecto inicial Estibaliz Rodríguez de Yurre*

Después de comentar el antiguo diseño con el director del proyecto se decidió hacer algunos cambios en la estructura, ya que no disponíamos de la base de datos antigua entre otras cosas. En la sección de trabajo realizado se detallará la nueva base de datos en más profundidad.

4.3 Puesta en producción

La puesta en producción de esta aplicación se ha realizado junto al alumno Gaizka Valle Zabala, la autora y con la colaboración del docente Pablo Gonzaléz Nalda. Se ha seguido un proceso de varios pasos.

El primer paso ha sido preparar el entorno de producción en *Proxmox*, un software para almacenar máquinas virtuales. Para ello se ha instalado *Node.js, Docker* y una base de datos *MySQL* en la máquina virtual (Ubuntu).

El segundo paso seguido ha sido subir los archivos de la aplicación a un directorio en la máquina virtual. Se añaden en un directorio aparte para poder actualizar la aplicación automáticamente con cada cambio realizado.

El tercer y último paso es configurar los archivos de *Docker* con todas las dependencias necesarias dentro del proyecto, así como enlazando con la base de datos creada en la máquina virtual.

En estos archivos de *Docker* es el lugar donde indicaremos que se vuelva a subir la aplicación cada vez que se realice un cambio el directorio del proyecto.

Cabe destacar que es necesario tener 2 versiones del proyecto, ya que las redirecciones cambian dentro de la máquina virtual respecto al entorno de desarrollo, teniendo que añadir siempre “/ramses” delante de cada dirección en el código.

Se puede ver la aplicación corriendo en el entorno de producción en la siguiente dirección:

https://lsi.vc.ehu.eus/ramses/

Capítulo 5. Trabajo realizado

En este capítulo se detallará el trabajo realizado en la aplicación mediante el uso de las herramientas y metodologías descritas en los apartados anteriores.

El primer problema a resolver fue la puesta en producción de la aplicación utilizando *Proxmox y Docker*. Como ya he comentado anteriormente esta labor fue desarrollada junto a mi compañero de proyecto Gaizka Valle Zabala.

Una vez en marcha el entorno de producción se dividieron las especificaciones y se comenzó la primera fase de implementación de manera individual.

5.1 Conversión entre JSON, JFLAP y BD

Este apartado consiste en poder dibujar el autómata pertinente y más tarde poder descargarlo tanto en *JSON* como en *JFLAP*. Si es justificado es posible que en las transformaciones se pierda información o no sea del todo equivalente (por ejemplo, la posición gráfica de los elementos o la información asociada a los ficheros) [10] .

Para ello se diseñaron funciones de conversión entre *XML y JSON*. Estos métodos realizan la conversión y descargan el autómata en el dispositivo pertinente. Se han integrado botones de descarga de *JSON y XML*. Estos botones se muestran en la Ilustración 28, señalados en color rojo.

Ilustración 28: *Conversión entre XML y JSON*

5.2 Gestión de idiomas

RAMSÉS ha integrado el potencial de utilizar cualquier idioma que se defina como soportado. Se han desarrollado mecanismos adecuados para que añadir un idioma nuevo sea solamente una cuestión de añadir la traducción de los textos a partir de las etiquetas existentes en otros idiomas [28] .

Se han integrado tres botones, uno por cada idioma. Se han integrado los idiomas: español, francés e inglés. Se ha creado un diccionario con las palabras que deben cambiar y una función que

realiza el cambio. En las siguientes ilustraciones podemos ver los botones que permiten el cambio de idioma y su resultado.

En las dos primeras ilustraciones (Ilustración 30 e Ilustración 29) podemos observar cómo los datos del *login* cambian según el idioma que elijamos.

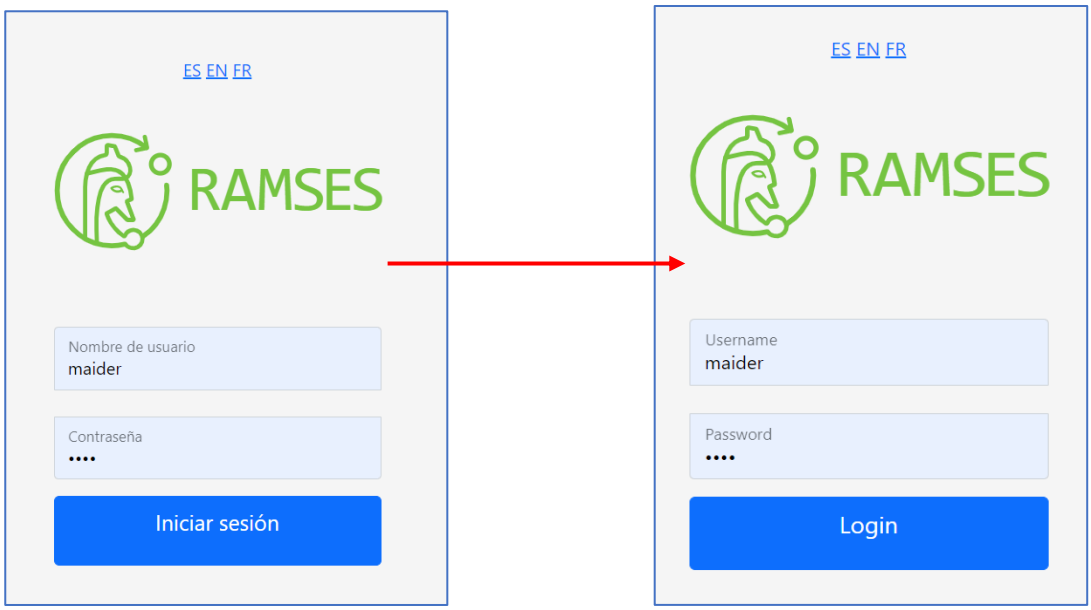

Ilustración 30: *Muestra cambio de idioma (I)*

Ilustración 29*: Muestra cambio de idioma (II)*

En las siguientes ilustraciones, Ilustración 31 e Ilustración 32, podemos observar como la información de carga y creación de autómatas también cambia según el idioma escogido.

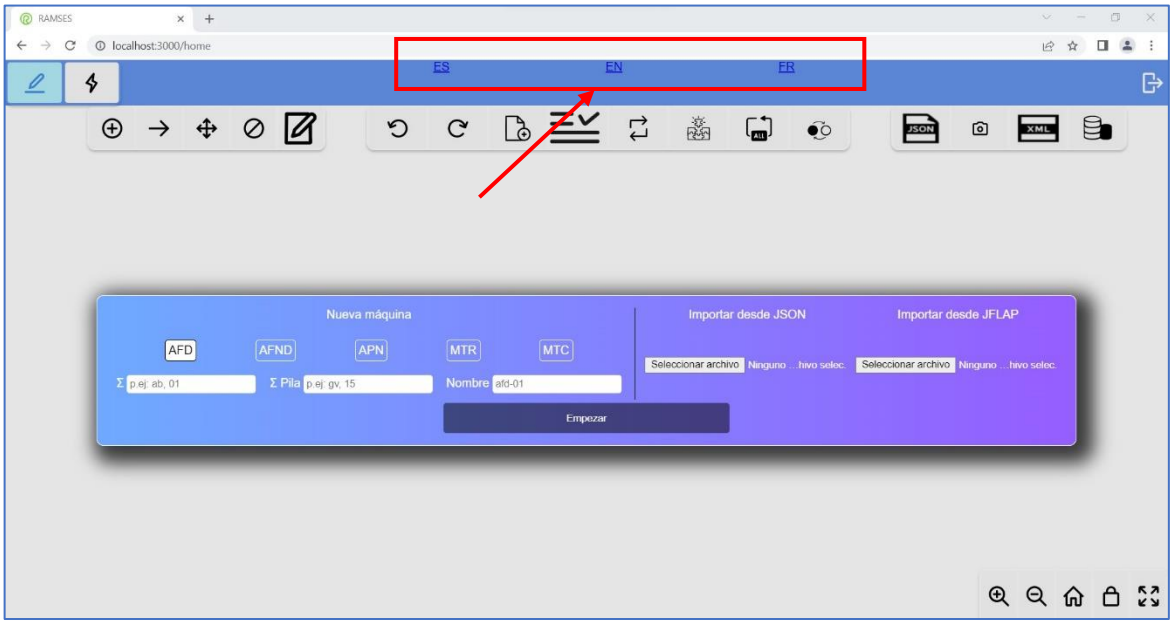

Ilustración 31: *Cambio idioma*

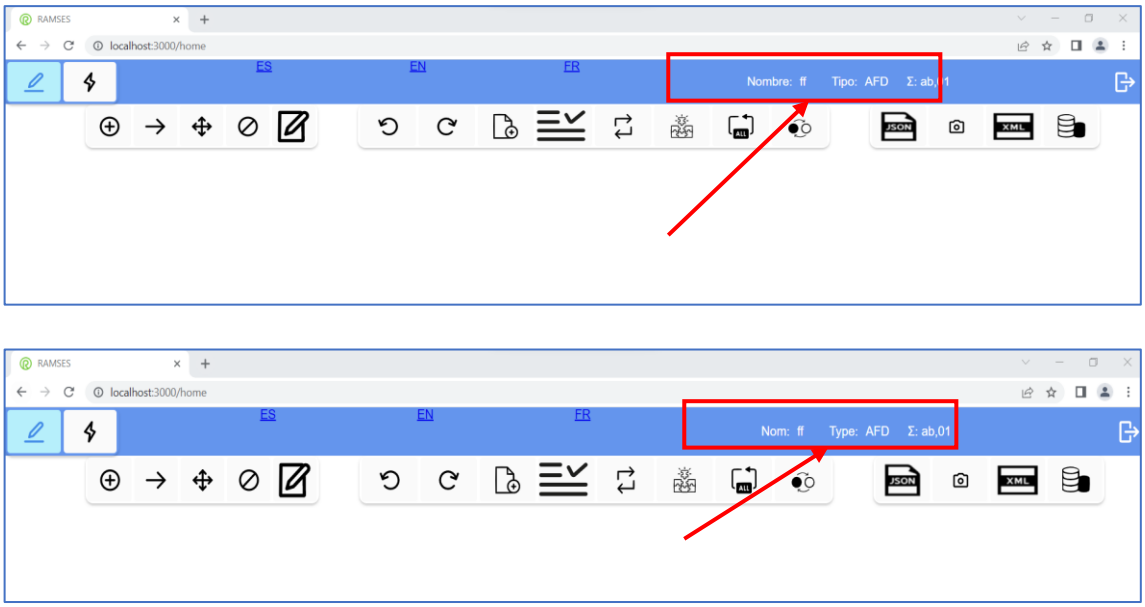

***Ilustración 32**: Cambio de idioma realizado*

5.3 Importación y exportación de autómatas

Respecto a la exportación de autómatas se ha comentado anteriormente que gracias a funciones de conversión es posible exportar el autómata tanto en *XML* como en *JSON*.

En cuanto a la importación RAMSÉS dispone de una sección de importación de autómatas. Esta sección la encontramos en la parte derecha de la página home.html, como se puede observar en la ilustración 33.

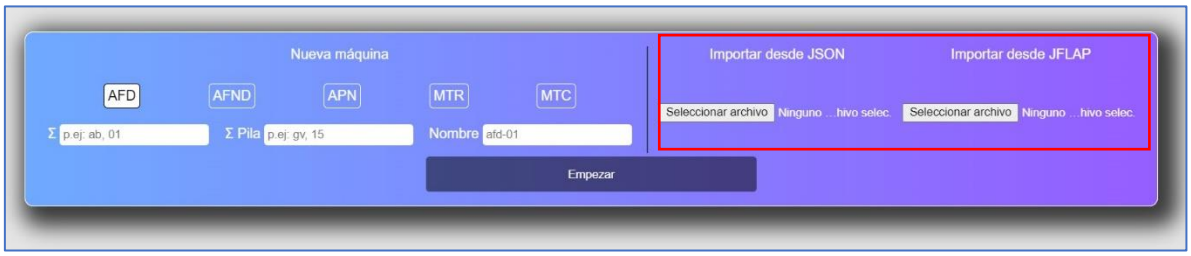

***Ilustración 33**: Importación de autómatas en XML/JSON*

Los botones de “Seleccionar archivo” permiten escoger un autómata desde el explorador de archivos. Como ya se ha comentado este autómata puede ser tanto *XML* como *JSON*, pero será imprescindible importar cada uno en su lugar. Esto lo podemos observar en la ilustración 34.

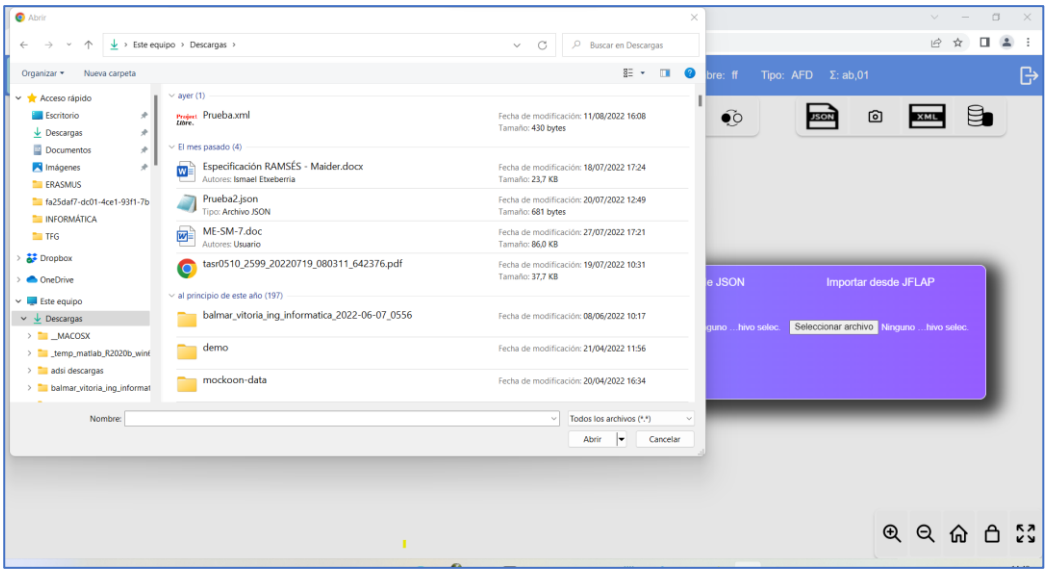

Ilustración 34: *Importación desde explorador de archivos*

Esto último es una excepción controlada, en caso de intentar subir un *JSON* en el apartado *XML* saltará el aviso de extensión del fichero no soportada. Esto se puede observar en la ilustración 35.

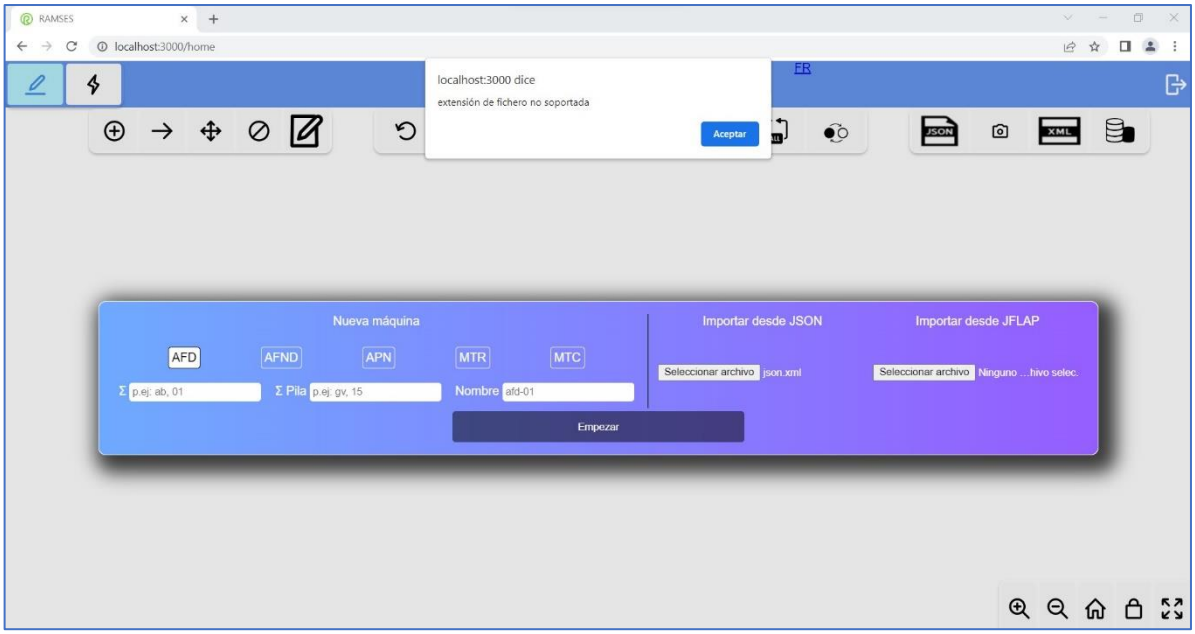

Ilustración 35: *Contenido no soportado en caso de equivocación*

En el caso de introducir un autómata valido en el correspondiente lugar y pulsar “Empezar” se mostrará de la manera que vemos en la *Ilustración 36.*

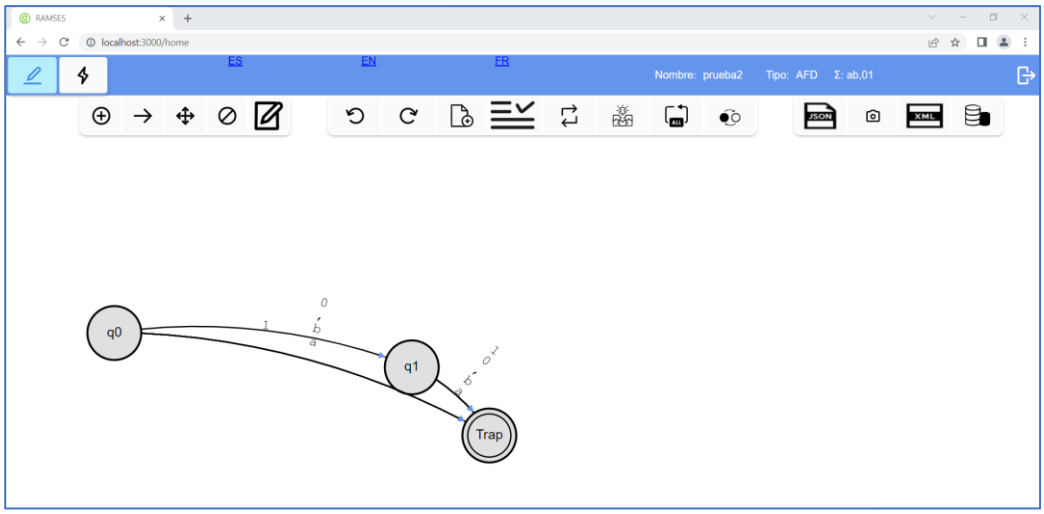

***Ilustración 36**: Visualización de autómata correctamente*

5.4 Monitorización

Se ha introducido un cronometro manual como modelo de funcionamiento de la monitorización del tiempo dedicado a cada actividad. Este cronometro podrá automatizarse en el caso de que en el futuro se implemente una corrección automática de los autómatas. Podemos ver el cronometro en la Ilustración 37.

***Ilustración 37:** Versión inicial monitorización*

5.5 Base de datos

Como se ha comentado con anterioridad, después de estudiar el diagrama entidad relación de la primera versión de RAMSÉS, se decidió realizar algunos cambios. Después de múltiples modificaciones el diagrama queda como se puede observar en la Ilustración 38.

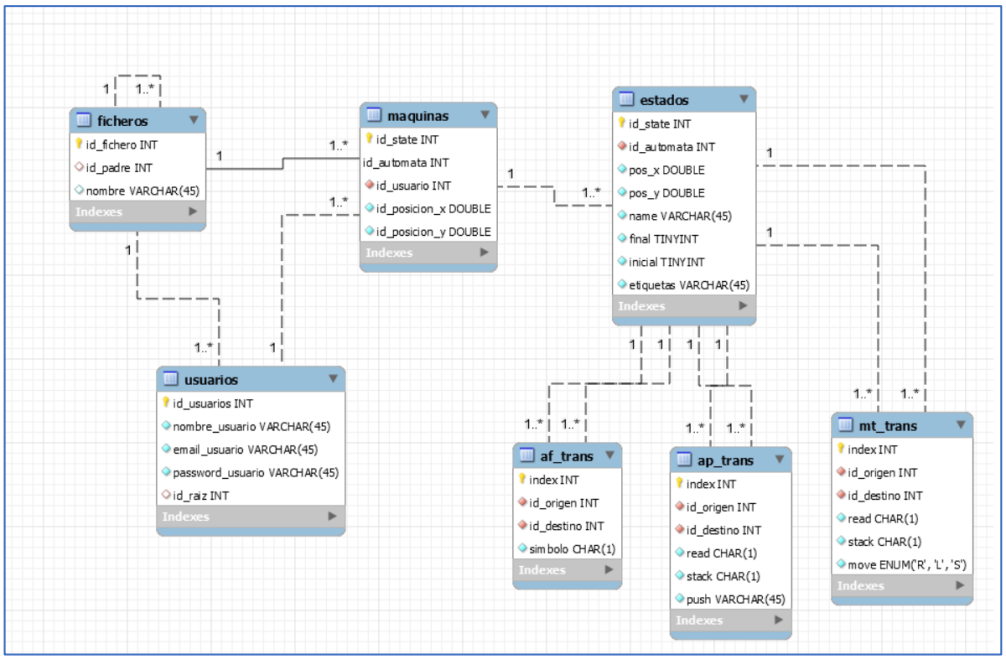

Ilustración 38: *Diagrama ER final*

La idea inicial era utilizar una *Adjacency List*[6], que se puede implementar en *MySQL* y es la más adecuada para cuando la jerarquía es actualizada frecuentemente. Después de comentar esta idea con el director del proyecto se decidió que, en vez de hacer uso de la idea inicial, la base de datos podía realizarse con una nueva tabla, llamada *ficheros* que sea una especie de "raíz" de la base de datos.

La idea principal de este diseño es que cada máquina tenga sus estados y sus transiciones, pero que además contenga un **identificador de usuario** para poder crear una especie de jerarquía en la clase fichero. Para entender este diagrama entidad relación debemos estudiar las tablas una por una.

En primer lugar, tenemos la tabla *fichero*. La tabla fichero contiene un **id_fichero** como clave primaria, un **id_padre** que crea una dependencia cíclica para la jerarquía y el **nombre** del fichero. Esta tabla está relacionada con las tablas *maquinas* y *usuarios*. Un fichero contiene uno o más usuarios y una o más máquinas.

La tabla *usuario* contiene el **id_usuarios, nombre, email, password y el id_raiz**. Los cuatro primeros atributos contienen la información del usuario.

El **id_raiz**, en cambio, es la clave foránea para conectar la tabla usuarios con las claves primarias de ficheros y máquinas.

La tabla *maquinas* contiene **id_state, id_automata, id_usuario, id_posicion_x e id_posicion_y**. Todos estos atributos son información de la máquina. La tabla *maquina* está

[6] Adjacency list. Ver en https://www.mysqltutorial.org/mysql-adjacency-list-tree/

relacionada con las tablas *fichero* y *estados*. Una o más máquinas contienen un fichero, una o más máquinas contienen un usuario y una maquina tiene uno o más estados.

La tabla *estados* contiene **id_state** como clave primaria, **id_automata** como clave foránea que conecta dicha tabla con la tabla *maquinas* y con las tablas de las *transiciones*. También posee las **posiciones** del estado, el **nombre**, dos valores booleanos para saber si es i**nicial o final** y una cadena de **etiquetas** para información extra. Uno o más estados pueden contener una máquina y una máquina puede contener una o más transiciones.

Las tablas de las transiciones cuentan con un **índex**, dos **ids** (uno de origen y otro de destino), sus **símbolos**, en el caso de que sea un autómata con pila o una máquina de Turing sus correspondientes **read, stack y push/move**. Un estado puede contener una o más transiciones.

El objetivo final sería que al crear o importar un autómata se pueda guardar en la base de datos pertinente, teniendo cada usuario *logueado* sus propios autómatas guardados. Esta tarea aún está a desarrollar ya que aún pueden surgir cambios en el diagrama entidad relación.

Se ha implementado un botón en la esquina superior derecha donde en un futuro el autómata completo se podrá subir a la base de datos. Lo podemos ver en la Ilustración 39.

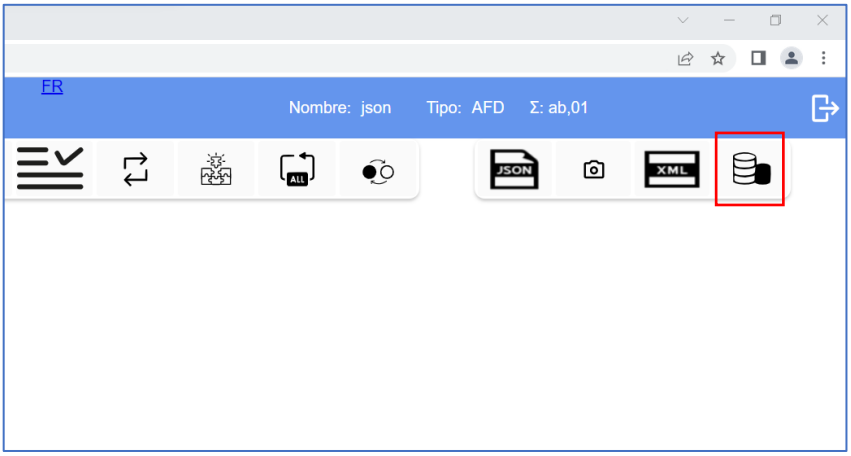

Ilustración 39: *Botón inserción en la base de datos*

5.6 Dificultades y Errores

En este apartado se describen las dificultades encontradas a la hora del desarrollo de la aplicación.

5.6.1. Conversión entre JSON Y JFLAP:

La primera tarea a realizar fue la conversión entre *JSON, JFLAP*, BD. Para ello se realizaron múltiples búsquedas entre libros, artículos, webs, etc. También se indagó en el código de la versión inicial de RAMSÉS, para que en la medida de lo posible el estilo de programación sea similar.

Hubo ciertas dificultades ya que no se podía acceder a los datos del autómata creado, ya que esa información se encontraba en otro archivo js. Tras múltiples intentos se consiguió recuperar la información. El problema fue que se intentó transformar el autómata antes del método *download*. Se solucionó añadiendo la conversión a dicho método. Esto se puede observar en la captura de código de la Ilustración 40.

```
var dataStr = "data:text/xml;charset=utf-8," + encodeURIComponent(this.chart.toDownload2());
console.log(dataStr);
var downloadAnchorNode = document.createElement('a');
downloadAnchorNode.setAttribute("href", dataStr);
downloadAnchorNode.setAttribute("download", filename2.concat(".xml"));
document.body.appendChild(downloadAnchorNode);
downloadAnchorNode.click();
downloadAnchorNode.remove();
```

Ilustración 40: *Corrección error I*

A continuación, se empezó la conversión del autómata, el cual estaba en forma de JSON. Se crearon los métodos de conversión entre XML y JSON sin ningún problema añadido.

5.6.2. Gestión página multi-idioma:

En la realización de la gestión de idiomas, tras realizar múltiples búsquedas se encontró una forma para la realización de este. En las páginas *HTML* funcionaba correctamente, pero en el cuadro de dialogo para la creación o importación de un autómata ha sido necesario añadir una propiedad al elemento que realizaba la traducción. Esto se puede observar en la captura de código de la Ilustración 41.

```
    <button-array id='modeButtons'></button-array>

    <a data-lang="es" href="javascript:translate(es)">ES</a>
    <a data-lang="en" href="javascript:translate(en)">EN</a>
    <a data-lang="fr" href="javascript:translate(fr)">FR</a>

    <div id="machine-info"></div>
    <button id="b-logout" type="click" onclick="window.location='/logout';">Logout</button>

</div>
<div id="menus">
```

Ilustración 41: *Muestra error II*

5.6.1. Importación de autómatas:

A la hora de la importación de los autómatas se trató de utilizar la conversión entre *XML* y *JSON*. Después de la creación del autómata en RAMSÉS los datos que se obtenían estaban en forma de cadena. Por ello que se tuvo que convertir la cadena *XML* en un objeto *JSON* para poder acceder a la información como *type, sigma, states,* etc. Después de varios cambios se consiguió acceder a prácticamente toda la información del autómata, toda menos la parte de "*states*".

Más tarde se consiguió acceder a "*states*" aunque no se consiguió acceder a la información dentro de este. En las siguientes ilustraciones se puede apreciar cual era el resultado obtenido en ese punto y el resultado deseado. Esto se puede observar en las capturas de código de las Ilustraciones 42 y 43.

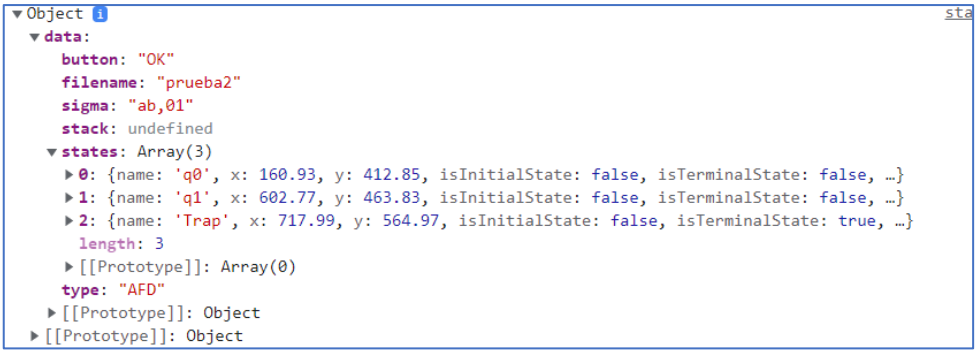

***Ilustración 42**: Resultado deseado*

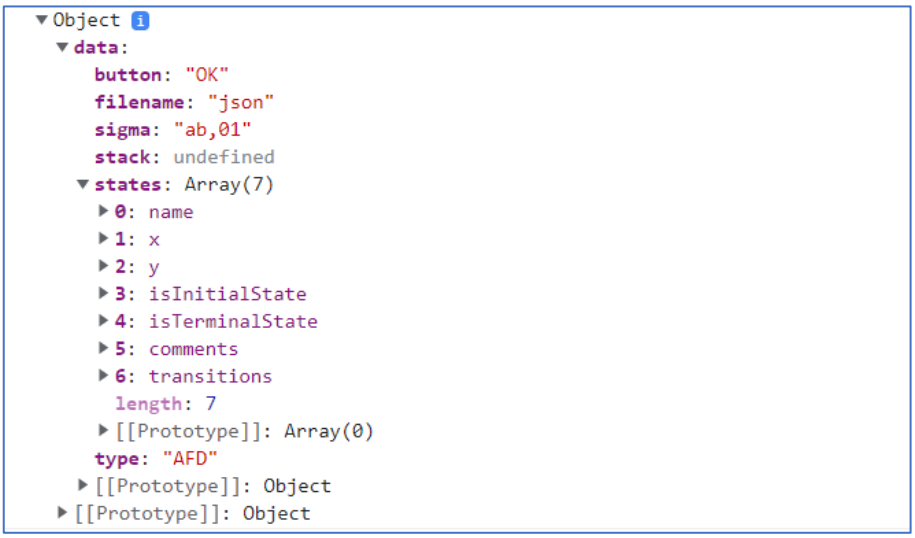

***Ilustración 43**: Resultado obtenido en ese punto*

Tras varias búsquedas y pruebas se consiguió subsanar el error y visualizar correctamente cada uno de los elementos de los estados, para posteriormente enviarla para la visualización del autómata en RAMSÉS.

En lugar de convertir los datos en *JSON* se optó por extraer directamente la información del *XML* como un objeto que después se pueda visualizar. Esto se puede observar en las Ilustraciones 44 y 45. Después de subsanar todos los errores RAMSÉS funciona correctamente.

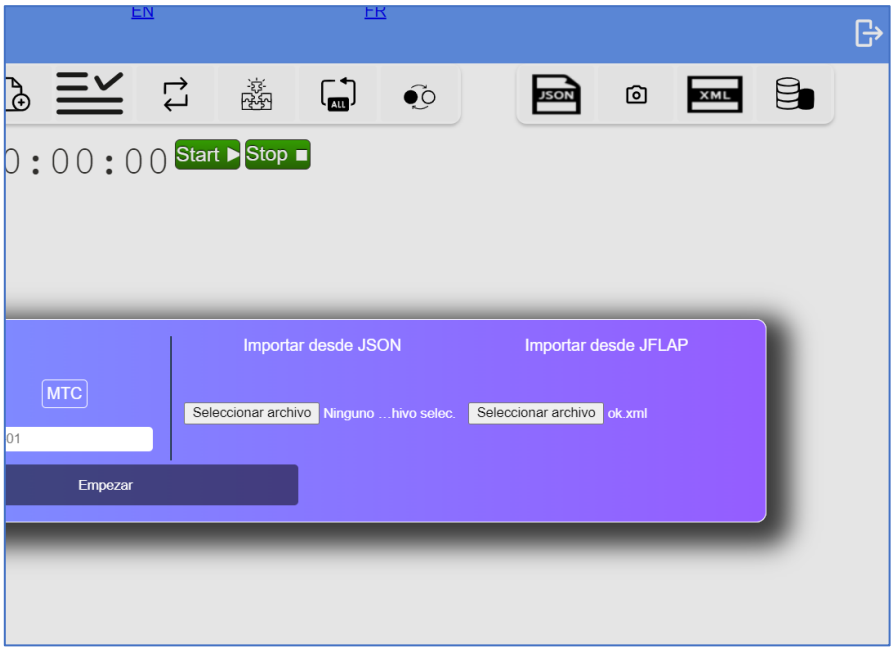

***Ilustración 44**: Importación en XML (I)*

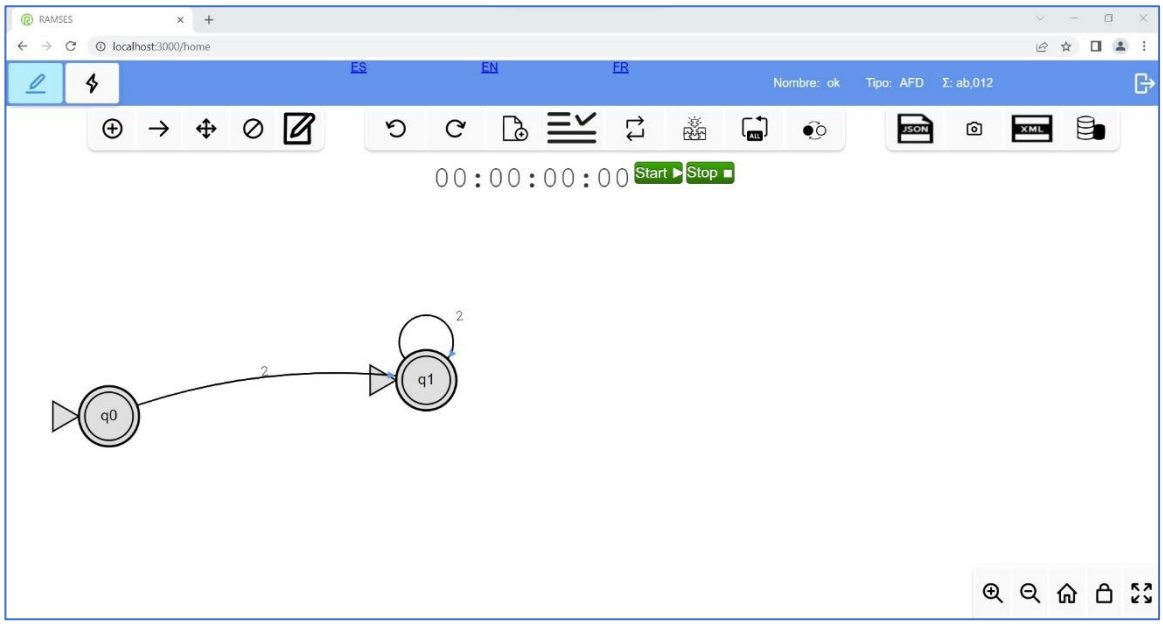

***Ilustración 45**: Importación en XML (II)*

Capítulo 6. Análisis final

6.1 Conclusión

Este proyecto es una continuación de un proyecto anterior en el que se creó una aplicación web que pudiera sustituir a la herramienta JFLAP (*Java Formal Languages and Automata Package)*. Esta aplicación fue creada por la antigua alumna, Estibaliz Rodríguez Yurre, como software de apoyo para los docentes de la asignatura Lenguajes, Computación y Sistemas Inteligentes (LCSI) en el Grado de Ingeniería Informática de Gestión y Sistemas de Información.

Este proyecto ha sido realizado en paralelo al alumno Gaizka Valle Zabala, que se ha encargado de otros apartados. También se ha llevado a cabo con la ayuda del profesor beneficiario de este software y director del proyecto. Se ha realizado una ampliación de la herramienta RAMSÉS consiguiendo así que en un futuro sea posible la creación y corrección de todo tipo de autómatas en varios formatos, así como su almacenamiento por autor.

Se ha realizado un análisis del conjunto global de especificaciones y se han implementado las funcionalidades que se han considerado más útiles o importantes con un resultado satisfactorio.

RAMSÉS ha supuesto un gran aprendizaje en herramientas y tecnologías conocidas y también desconocidas. Ha supuesto una inmersión en el mundo de JavaScript para crear un proyecto de esta magnitud.

Por otro lado, el tener que interactuar con el cliente para lograr subsanar los cambios en las especificaciones han supuesto un gran aprendizaje de cara al mundo laboral aprendiendo a cómo adaptarse cuando sea necesario.

Tras la finalización del proyecto y teniendo en cuenta que se ha desarrollado fuera del ámbito de una empresa, con el desarrollo de este parte realizado por una única persona, puede considerarse el proyecto como exitoso.

6.2 Líneas futuras

RAMSÉS es una aplicación web con mucho potencial para crecer, da pie a implementar nuevas funcionalidades que pueden ser muy útiles en el futuro.

El objetivo a largo plazo es poder integrar esta aplicación para que el alumnado pueda usarla como material complementario. A día de hoy RAMSÉS está instalado en el servidor del departamento de LSI siendo accesible desde la Web. Una idea para el futuro podría ser integrarlo en la plataforma

eGela para que el alumnado pueda hacer uso de él con sus credenciales. Esto permitiría incorporar diferentes tipos o roles para los usuarios.

La idea es que en el futuro se puedan añadir nuevas funcionalidades, o modificar las ya existentes en función de la respuesta del usuario final real, es decir del alumnado y el profesor de la asignatura. Algunas de estas funcionalidades futuras fueron recogidas en el análisis inicial del proyecto, y descartadas para el desarrollo actual. Por ejemplo, realizar una monitorización del tiempo al realizar los ejercicios. En este momento RAMSÉS dispone de un contador manual que en un futuro podría automatizarse al integrar una funcionalidad de corrección de los autómatas.

Además, al inicio del proyecto se habló de que podría ser útil migrar la aplicación a *Vue.js*, *React* o *Angular*. En particular, *React,* ayuda a crear interfaces de usuario interactivas de forma sencilla. Diseña vistas simples para cada estado en la aplicación, y se encargará de actualizar y renderizar de manera eficiente los componentes correctos cuando los datos cambien. Por todo ello *React* sería una buena opción a futuro [25] .

Bibliografía

Libros, material académico y sitios web

[1] Atlassian. (s.f.). *Explicación de como realizar un diagrama de Gantt*. Obtenido de https://www.atlassian.com/es/agile/project-management/gantt-chart

[2] Attlasian. (s.f.). *¿ Qué es la integración continua?* Obtenido de https://www.atlassian.com/es/continuous-delivery/continuous-integration

[3] BOE. (s.f.). *«BOE» núm. 128. Resolución de 17 de mayo de 2022*. Obtenido de https://www.boe.es/diario_boe/txt.php?id=BOE-A-2022-8783

[4] Brena, R. (2003). *Autómatas y lenguajes: Un enfoque de diseño.* ITESM.

[5] Cisco. (s.f.). *Introducción a cisco anyconnect para vpn*. Obtenido de https://www.cisco.com/c/dam/global/es_mx/products/pdfs/cisco-anyconnect-secure-esp.pdf

[6] DesarrolloWeb. (s.f.). *Información sobre HTML*. Obtenido de https://desarrolloweb.com/home/html

[7] Docker. (s.f.). *Explicación de software Docker en Wikipedia*. Obtenido de https://es.wikipedia.org/wiki/Docker_(software)

[8] DreamHost. (s.f.). *Configuración de software PuTTY*. Obtenido de [8] https://help.dreamhost.com/hc/es/articles/215464538-Configurar-PuTTY

[9] Etxeberria, I. (s.f.). *Apuntes de la asignatura de LCSI.* UPV/EHU.

[10] Friesen, J. (2016). *Java XML and JSON.* Apress.

[11] Git. (s.f.). *Descargar Git*. Obtenido de https://git-scm.com/downloads

[12] GitHub. (s.f.). *Sitio web GitHub*. Obtenido de https://github.com/

[13] HTML. (s.f.). *HTML EN W3SCHOOLS*. Obtenido de https://www.w3schools.com/html/

[14]Jflap.org. (s.f.). *Sitio web descarga JFLAP*. Obtenido de http://www.jflap.org/tutorial/

[15]JSON.org. (s.f.). *Información sobre JSON, JavaScript Object Notation*. Obtenido de https://www.json.org/json-es.html

[16]Lenguaje JS. (s.f.). *Información sobre como hacer uso de WebComponents*. Obtenido de https://lenguajejs.com/webcomponents/introduccion/que-son-webcomponents/

[17]mdn web docs. (s.f.). *Conceptos básicos sobre Flexbox*. Obtenido de https://developer.mozilla.org/es/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox

[18]mdn web docs. (s.f.). *Información de interés sobre CSS*. Obtenido de https://developer.mozilla.org/es/docs/Web/CSS

[19]mdn web docs. (s.f.). *Información de interés sobre Media Queries*. Obtenido de https://developer.mozilla.org/es/docs/Web/CSS/Media_Queries/Using_media_queries

[20]Microsoft. (s.f.). *¿ Qué es Git?* Obtenido de https://docs.microsoft.com/es-es/devops/develop/git/what-is-git

[21]MySql.com. (s.f.). *Instalación de MySQL*. Obtenido de https://dev.mysql.com/downloads/installer/

[22]Node.js. (s.f.). *Explicación de Node.js en Wikipedia*. Obtenido de https://es.wikipedia.org/wiki/Node.js

[23]ProjectLibre. (s.f.). *Introducción a ProjectLibre en Wikipedia*. Obtenido de https://es.wikipedia.org/wiki/ProjectLibre

[24]Proxmox. (s.f.). *Introducción al entorno virtual proxmox*. Obtenido de https://es.wikipedia.org/wiki/Proxmox_Virtual_Environment

[25]React. (s.f.). *Sitio web librería OpenSource React*. Obtenido de https://es.reactjs.org/

[26]Semantic System. (s.f.). *Cómo realizar una gestión de riesgos de un proyecto emergente*. Obtenido de https://www.semantic-systems.com/semantic-noticias/articulos-tecnologicos/gestion-de-riesgos-en-proyectos-de-implantacion-de-software/

[27]SQL. (s.f.). *Información sobre SQL en Wikipedia*. Obtenido de https://es.wikipedia.org/wiki/SQL

[28]StackOverflow. (s.f.). *Explicación de convertir una página web en página web multi-idioma*. Obtenido de https://stackoverflow.com/questions/42494867/i-want-to-translate-my-html-text

[29]SVG. (s.f.). *Información de uso de vectores gráficos escalables en Wikipedia*. Obtenido de https://es.wikipedia.org/wiki/Gr%C3%A1ficos_vectoriales_escalables

[30]Villa, E. R. (s.f.). *RAMSES: Entorno de Simulación de Máquinas Abstractas.*

[31]Visual Studio Code. (s.f.). *Información sobre herramienta Visual Studio Code en Wikipedia*. Obtenido de https://es.wikipedia.org/wiki/Visual_Studio_Code

[32]W3SCHOOLS. (s.f.). *Información general sobre W3SCHOOLS, sitio web para aprender tecnologías web en línes*. Obtenido de https://www.w3schools.com/

[33]XML. (s.f.). *XML en W3SCHOOLS*. Obtenido de https://www.w3schools.com/xml/xml_whatis.asp

Manuales y tutoriales

- Jflap.org. (s.f.). Obtenido de http://www.jflap.org/tutorial/
- W3SCHOOLS. (s.f.). Obtenido de https://www.w3schools.com/

Glosario

BD	base de datos
BOE	Boletín Oficial del Estado
CSS	Cascading styling sheets
HTML	HyperText Markup Language
IVA	Impuesto sobre valor añadido
JS	JavaScript
JSON	JavaScript Object Notation
MySQL	Sistema de gestión de bases de datos relacional
RAMSÉS	Recursive Abstract Machines Simulation EnvironmentS
SVG	Scalable Vector Graphics
TFG	Trabajo de fin de grado
VC	Visual Code
XML	eXtensible Markup Language (Lenguaje de marcado extensible)

Anexos

Anexo I – Manual de actualización de entorno de producción

El primer paso a realizar es conectarse al servidor de la escuela. Se puede mediante un *shell* remoto, por ejemplo, *PuTTY* y mediante FTP, por ejemplo, *FileZilla*. En el caso de *PuTTY* en la configuración añadimos como se puede observar en la Ilustración 46.

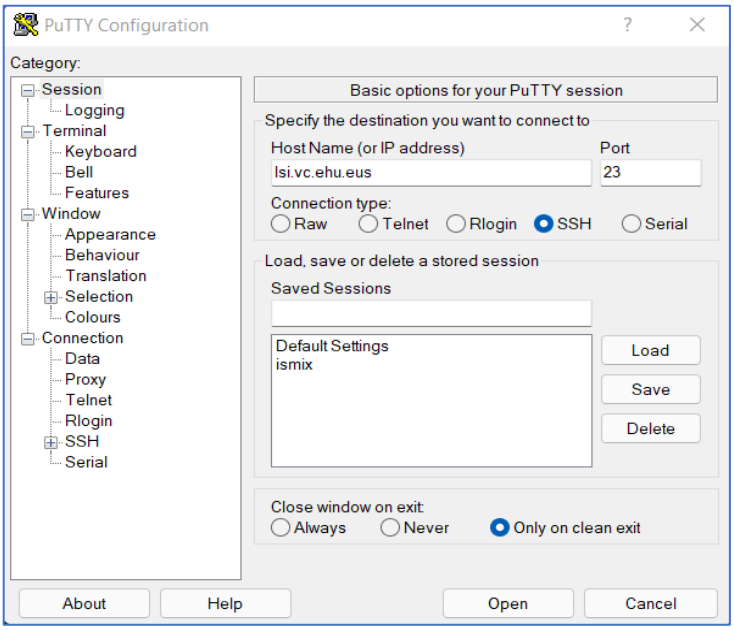

Ilustración 46: *Configuración de RÁMSES en PuTTY*

Respecto a cómo realizarlo en *FileZilla*, entramos en la pestaña "Gestor de sitios", al igual que hemos realizado anteriormente en el *PuTTY* introducimos el host (**lsi.vc.ehu.eus**), el usuario (**ramses**) y la contraseña. En este caso el protocolo a utilizar sería *SFTP*. Podemos ver la configuración en la Ilustración 47.

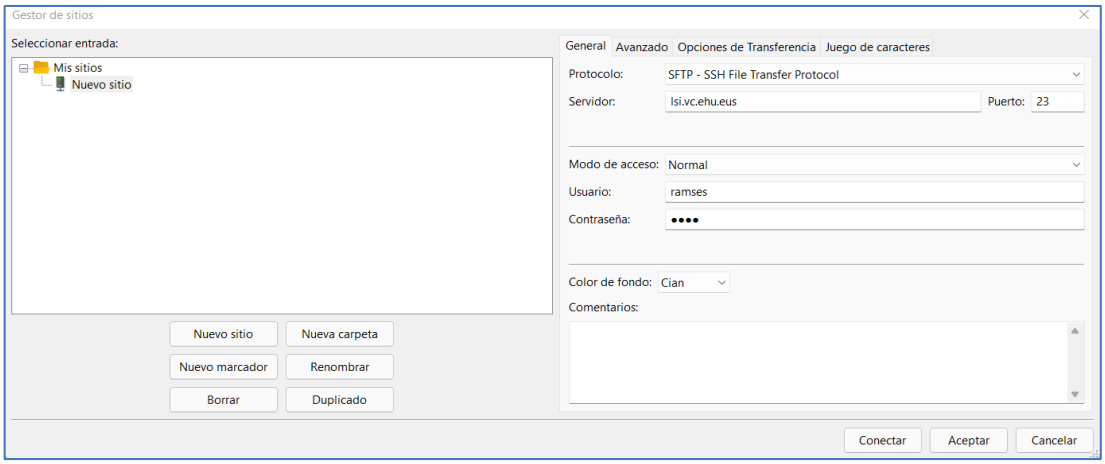

Ilustración 47: *Configuración de Ramsés en Filezilla*

El contenido de la máquina virtual en *FileZilla* se puede observar en la Ilustración 48.

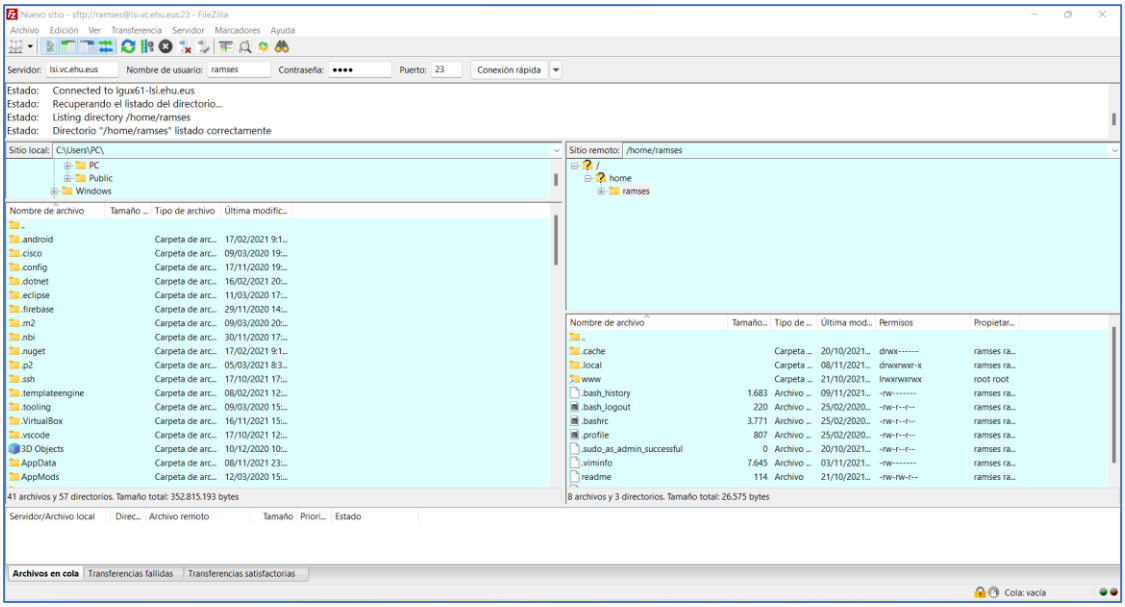

***Ilustración 48**: Visualización contenido de Ramsés en Filezilla*

Una vez conseguido el acceso, en *FileZilla* se buscan los archivos a sustituir y se arrastran las nuevas versiones, sobrescribiendo las anteriores.

En todos los sitios que se necesita redirección (como podría ser el *path* a una imagen o un cambio de página) hay que introducir ***/ramses/*** al inicio en la versión que se sube a la máquina virtual. Esto pasa mayormente en los *CSS*, *button-array.js* y el botón de volver al *login* en el *home* por lo cual hay que prestar mucha atención a que las redirecciones se hagan correctamente antes de subirlo ya que, si no, no se visualizará correctamente.

El otro archivo que tiene cambios y redirecciones en comparación a la versión de desarrollador para funcionar correctamente en el entorno preparado para producción es el *index.js* donde hay que tener especial cuidado si se sobrescribe.

Al subir mediante *FileZilla* los archivos modificados se actualizan solos, pero en el caso de cambiar parte de la configuración es necesario entrar en *PuTTY* y ejecutar los siguientes comandos:

```
$ docker-compose down
$ docker-compose up -d
```

Anexo II - Manual de instalación

El presente anexo es una versión ampliada y más actualizada del manual de instalación proporcionada en el Anexo 1 de la memoria del TFG de Estibaliz Rodríguez de Yurre.

En él se detallan los pasos que se han seguido, previos al desarrollo de la aplicación, para la instalación y puesta en marcha de software y servicios de terceros. Se especifica la instalación para el SO *Windows 11(64 bits).*

Para ejecutar la aplicación final es necesario contar con *Node.js* y *MySQL* instalados. Los módulos de terceros de *Node.js* se incluyen en el entregable de la aplicación, por lo que no es necesario volver a descargarlos, pero en caso de necesitarlos por algún error se instalan con uno de los siguientes comandos:

```
$ npm i
$ npm install
```

Instalación de Node.js

Descargar en instalador **(.msi)** desde [la página oficial de descargas de Node.js.](#) En las Ilustraciones 49 y 50 podemos observar el proceso de instalación.

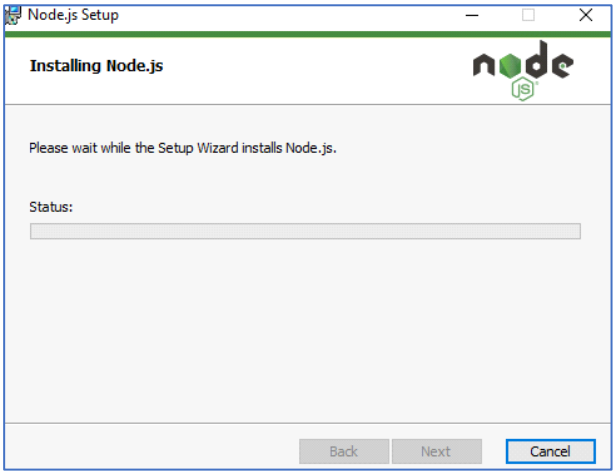

Ilustración 49: *Instalación Node.js*

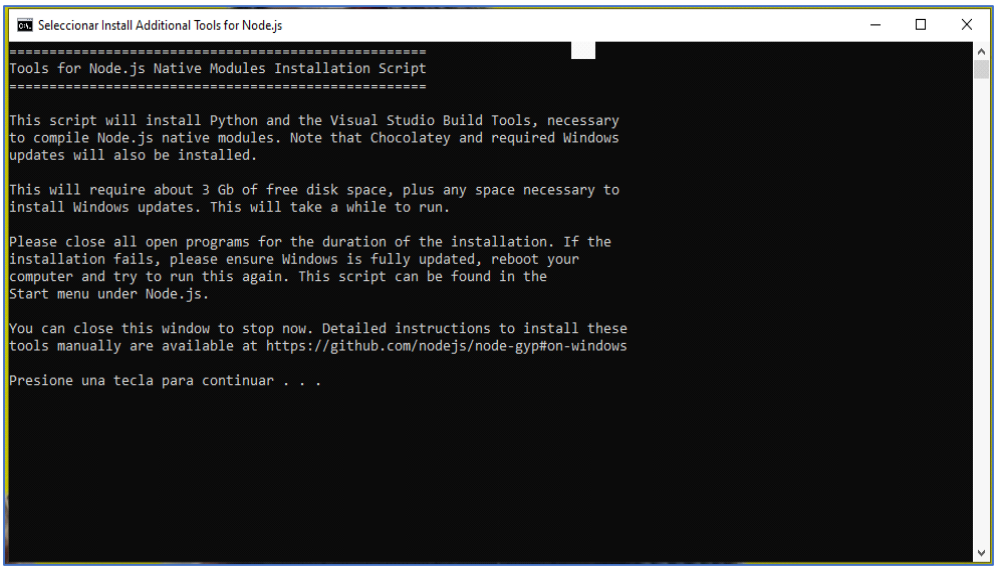

Ilustración 50: *Instalación herramientas adicionales*

A continuación, se comprueba que la instalación se ha realizado correctamente mediante los comandos *node* y *npm*, que devuelven sus correspondientes números de versiones. Podemos ver las versiones instaladas en la Ilustración 51.

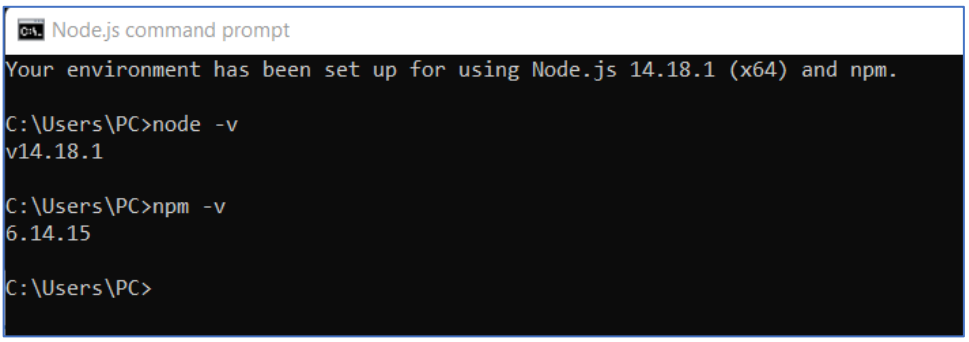

Ilustración 51: *Comprobación versiones node*

Módulo de terceros

Para este proyecto se han utilizado módulos (paquetes de funcionalidades) que se instalan mediante el comando ***npm install,*** que proporcionan las funcionalidades siguientes:

- **Express**: Proporciona la infraestructura para manejar peticiones GET/POST, seleccionar el puerto de conexión, y crear un sistema de *routing*.
- **Express-session:** Extensión de *Express* para la gestión de sesiones de usuario en el navegador.
- **Body-parser:** *Middleware* de *Express* que permite acceso al cuerpo de una petición POST (*payload*).
- **MySQL:** Permite conectarse a (y desconectarse de) una base de datos *MySQL* y realizar queries para obtener información de la misma.

Instalación de MySQL

Se ha instalado la última versión estable de *MySQL Community*, que incluye *MySQL Workbench* como interfaz gráfica para la gestión CRUD de la BD [21]. En la Ilustración 52 podemos observar el *MySQL Installer*.

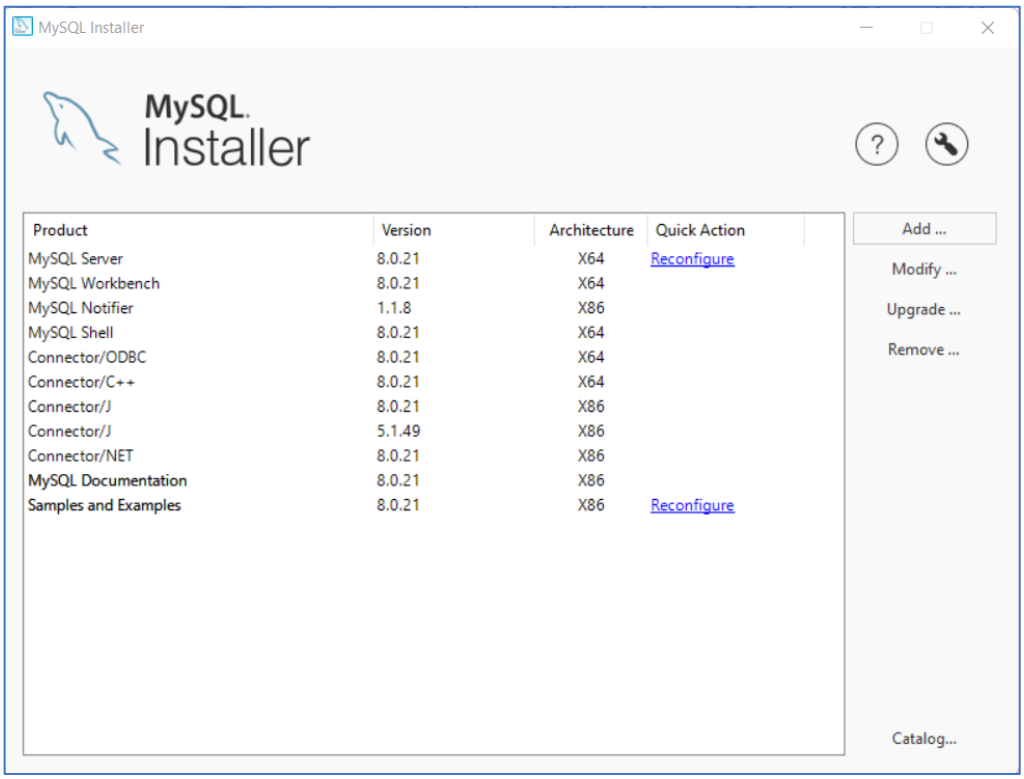

***Ilustración 52**: MySQL community installer.*

En este punto se debe crear una base de datos, y un usuario (nombre de usuario y contraseña) para acceder como administrador. Estas credenciales son las que se utilizarán para realizar la conexión a la BD desde el código de la aplicación.

Esto se realizará en el caso de no poseer la base de datos completa tras esta ampliación del proyecto. En las Ilustraciones 53 y 54 podemos observar el estado del contenido de la base de datos al comienzo y al final.

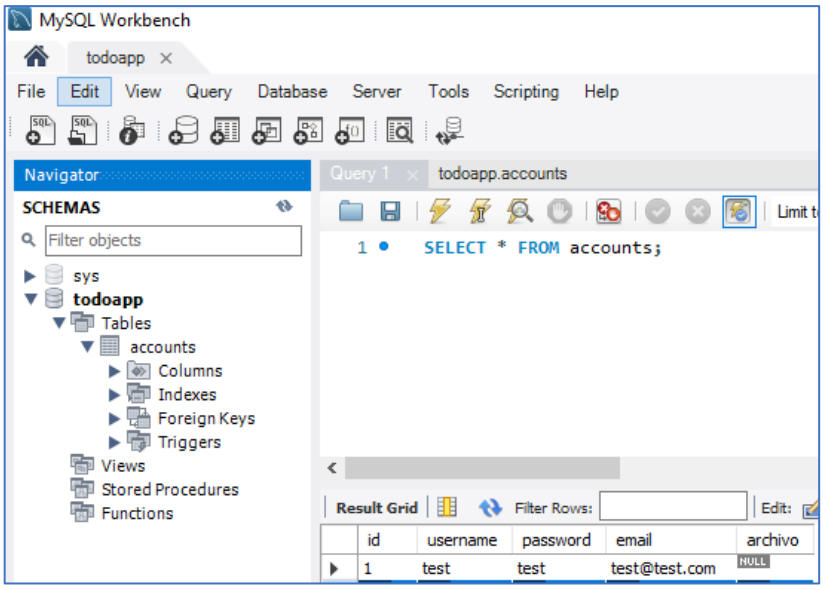

Ilustración 53: *MySQL workbench version simple*

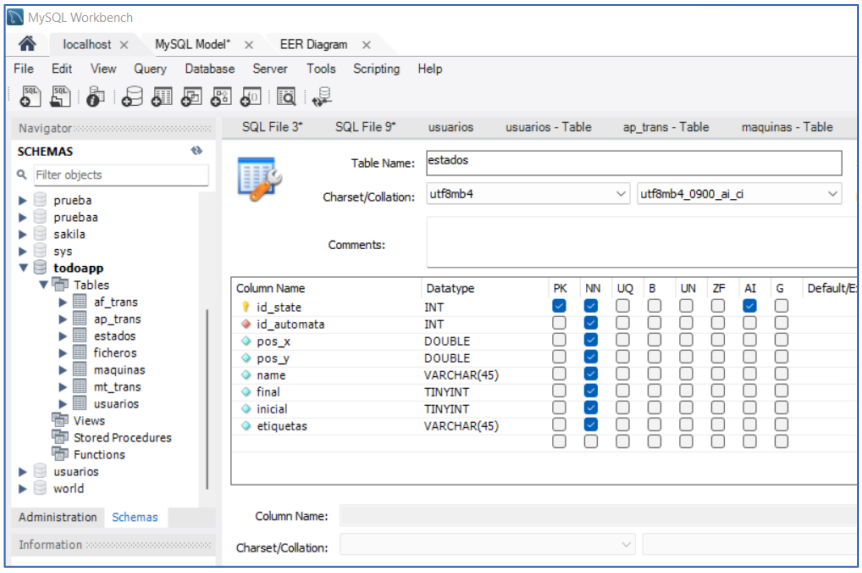

Ilustración 54: *MySQL WORKBENCH versión completa*

En este mismo documento hay un diagrama entidad relación para crear la base de datos, pero en los entregables del proyecto hay una copia llamada **ramses.sql** para su importación completa.

Instalación de Git

El objetivo de la instalación de git en *Windows* es la utilización de un repositorio en *GitHub* para el proyecto, ya que es necesaria la creación de una clave SSH [11]. En la ilustración 55 se puede observar el proceso de instalación en *Git*.

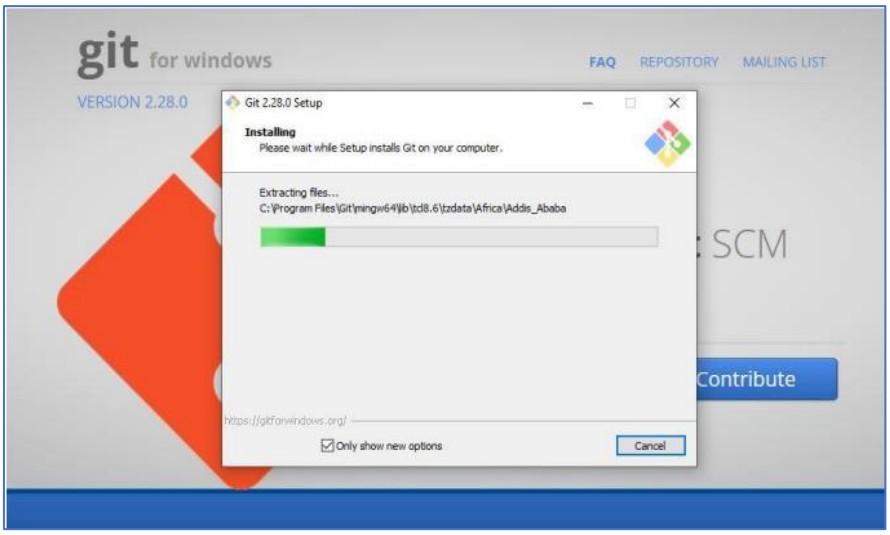

***Ilustración 55**: Instalación en Git*

Con *Git* se instalan varias aplicaciones distintas, pero para este proyecto hemos hecho uso de ***Git Bash*.**

Git Bash permite utilizar herramientas *Linux Bash* con *Git* en la línea de comandos.

Configuración de repositorio en GitHub

El primer paso es crearse una cuenta en *GitHub*, o en su defecto en el caso de poseer una iniciar sesión. A continuación, se genera una clave SSH con los siguientes comandos en la terminal:

```
$ ssh-keygen -t rsa -b 2048
```

Después ejecutaremos el siguiente comando para activar el agente de *ssh* consiguiendo así no tener que añadirla cada vez que encendemos el ordenador:

```
$ ssh-add
```

Podemos comprobar que se han generado correctamente con:

```
$ ssh-add -l
```

El último paso con respecto a la clave *SSH* es añadirla a nuestra cuenta de *GitHub* en *Settings* > SSH and GPG Keys.

En este caso el repositorio fue creado por mi compañero de proyecto, por lo que mediante una invitación conseguí acceso al repositorio ya que era privado [12]. La ilustración 56 muestra la notificación para colaborar en *GitHub*.

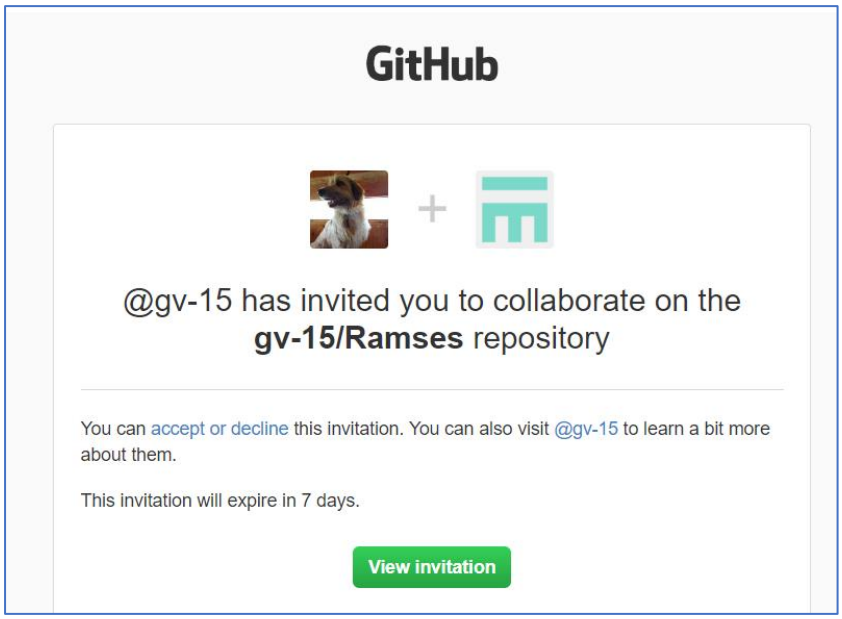

***Ilustración 56**: Invitación a repositorio de GitHub*

Al aceptar la invitación el creador ofrece los permisos de ver y editar en este caso. El repositorio de GitHub se ve de la manera que muestra la Ilustración 57.

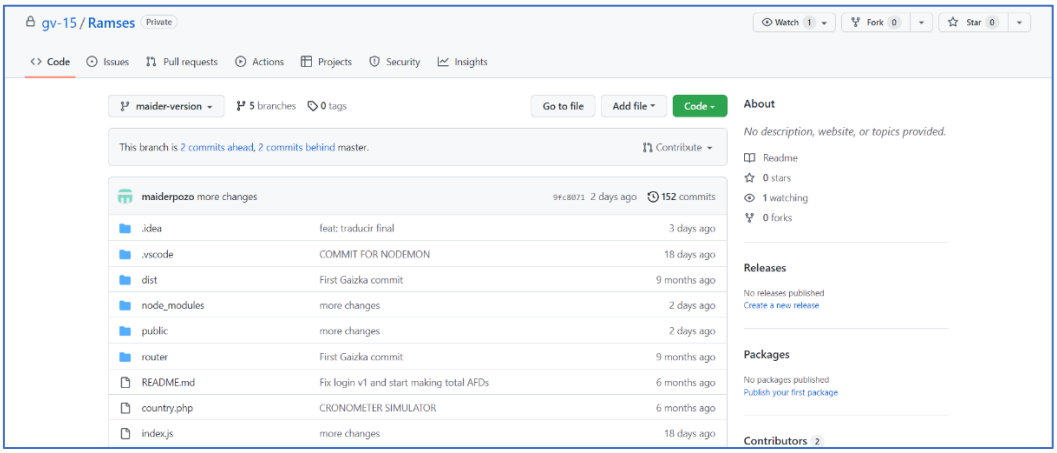

***Ilustración 57**: Repositorio de Ramsés*

Instalación de Visual Studio Code

Visual Studio Code, también conocido con *VS Code*, es el editor de código escogido para realizar el proyecto. Esta herramienta es gratuita y tiene varias funcionalidades muy útiles para nuestro proyecto. Se puede descargar desde la página oficial [7]. La Ilustración 58 muestra el proyecto en dicho editor.

[7] https://code.visualstudio.com/download

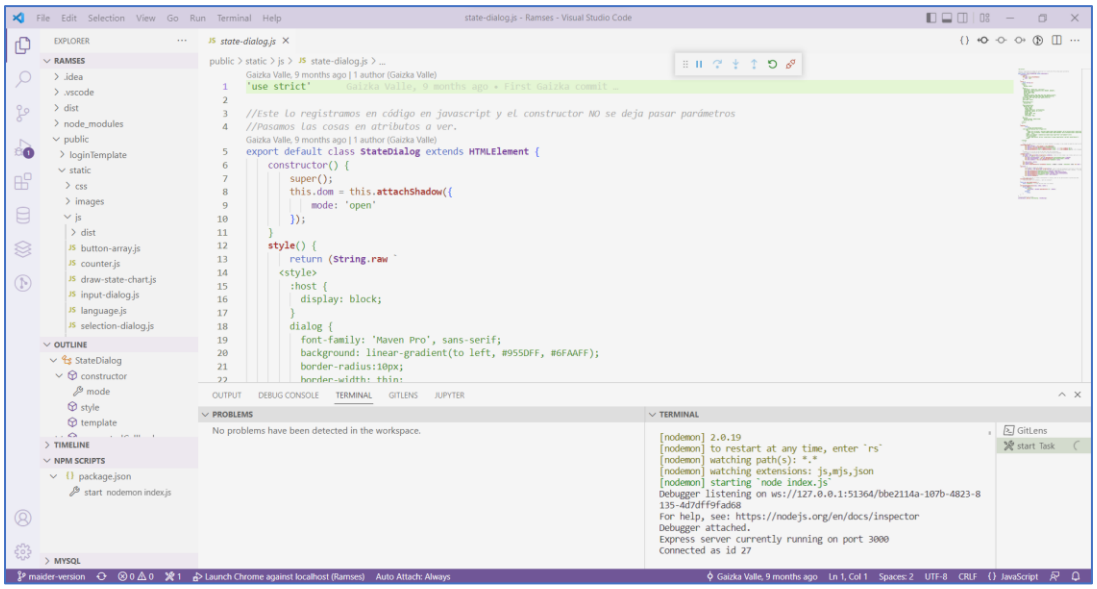

***Ilustración 58**: RAMSÉS en Visual Studio Code*

Cabe destacar que en el caso de tener configurado el proyecto con **ssh** (como es el caso de RAMSÉS), desde el mismo **GitHub** se puede clonar el repositorio directamente.

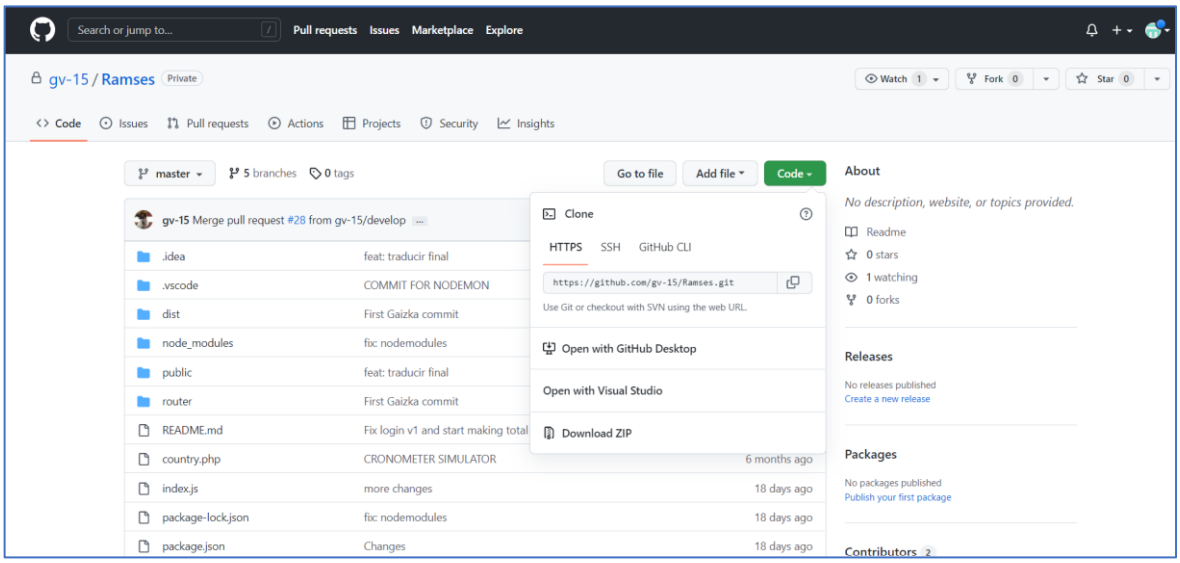

***Ilustración 59:** Clonación desde GitHub*

Agradecimientos

Por último, agradecer a la Universidad del País Vasco/ Euskal Herriko Unibertsitatea UPV/EHU el haberme dado la oportunidad de participar en este proyecto. También dar las gracias al director de este trabajo de fin de grado, por haberme instruido y ayudado a estructurar el trabajo. Y por supuesto, a Estibaliz Rodríguez de Yurre por dejarme participar en su proyecto, y Gaizka Valle por su colaboración.