



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

Draw Entity-Relation App



Presentado por Rubén Maté Iturriaga
en Universidad de Burgos — 7 de julio de 2024
Tutor: Jesús Manuel Maudes Raedo



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



D. Jesús Manuel Maudes Raedo, profesor del departamento de Ingeniería Informática, área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Rubén Maté Iturriaga, con DNI 71364920Z, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 7 de julio de 2024

Vº. Bº. del Tutor:

D. Jesús Manuel Maudes Raedo

Resumen

Este proyecto permite modelar diagramas entidad-relación en una aplicación web y exportarlos a un script SQL tras su validación. Es también un punto de partida para que futuros proyectos puedan expandir su funcionalidad con relaciones más complejas. La aplicación está desarrollada en React y utiliza la librería mxGraph para la creación y manipulación de los diagramas.

Descriptores

Modelado de bases de datos, diagramas entidad-relación, exportación a SQL, aplicación web, React, mxGraph.

Abstract

This project allows modeling entity-relationship diagrams in a web application and exporting them to an SQL script after validation. It's also a starting point for future projects that will add features and more complex relations. Tusehe application is developed in React and uses the mxGraph library for the creation and manipulation of the diagrams.

Keywords

Database modeling, entity-relationship diagrams, SQL export, web application, React, mxGraph.

Índice general

Índice general	iii
Índice de figuras	v
Índice de tablas	vii
1. Introducción	1
1.1. Estructura de la memoria	2
1.2. Estructura de los anexos	2
1.3. Enlaces	3
2. Objetivos del proyecto	5
2.1. Objetivos generales	5
2.2. Objetivos técnicos	5
3. Conceptos teóricos	7
3.1. Aplicación web	7
3.2. Desarrollo web moderno	8
3.3. UI y UX	9
3.4. Linter	9
3.5. Integración continua	9
3.6. Despliegue continuo	10
3.7. Validación y pruebas	10
4. Técnicas y herramientas	11
4.1. mxGraph	11
4.2. React	11

4.3. Material UI	11
4.4. Biome	11
4.5. Herramientas utilizadas para validación y pruebas	12
4.6. Herramientas utilizadas para integración continua	12
4.7. Despliegue continuo	13
4.8. Git	13
4.9. GitHub	14
5. Aspectos relevantes del desarrollo del proyecto	15
5.1. Metodologías	15
5.2. Desarrollo del proyecto	19
6. Trabajos relacionados	53
6.1. Draw.io	53
6.2. Excalidraw	54
6.3. Comparativa frente a alternativas	55
7. Conclusiones y Líneas de trabajo futuras	57
7.1. Conclusiones	57
7.2. Líneas de trabajo futuras	58
Bibliografía	61

Índice de figuras

5.1. Milestones en GitHub	16
5.2. Labels en GitHub	17
5.3. Ejecución automática de tests unitarios	19
5.4. Inicialización de mxGraph	21
5.5. HTML renderizado por el componente de React DiagramEditor.js	21
5.6. Aplicación tras implementar la posibilidad de añadir entidades .	22
5.7. Primera versión de la representación interna del diagrama E-R .	23
5.8. Botón contextual para entidades	24
5.9. Añadir atributos a una entidad	25
5.10. Muestra de una ejecución de Biome con errores	26
5.11. Muestra de una comprobación previa a un commit con Lefthook	26
5.12. Ejecución de pruebas unitarias	27
5.13. Ejecución de pruebas End to End	28
5.14. Ejecución de las diferentes GitHub Actions en nuestro proyecto	29
5.15. Atributo clave subrayado con elipse no coloreada y lado no dirigido	30
5.16. Botón contextual para convertir un atributo que no es clave en clave	30
5.17. Botones contextuales al seleccionar una relación	31
5.18. Interfaz de configuración de los lados de una relación	32
5.19. Interfaz de configuración de las cardinalidades de los lados de una relación	33
5.20. Relaciones en el canvas	34
5.21. Diagrama interno guardando relaciones	36
5.22. Interfaz contextual para una relación N:M	37
5.23. Función de validación del diagrama con diagnósticos (Parte 1) .	39
5.24. Función de validación del diagrama con diagnósticos (Parte 2) .	40
5.25. Flujo para procesar la estructura interna y generar el script SQL	41

5.26. Procesado de relaciones (Parte 1)	42
5.27. Procesado de relaciones (Parte 2)	43
5.28. Procesado de tablas y merge	44
5.29. Procesado de tablas finales para eliminar acentos y evitar posibles atributos repetidos.	45
5.30. Generación del string SQL final	45
5.31. Ejemplo de script SQL con ALTER TABLE al final para crear las relaciones foráneas.	46
5.32. Diagrama interno guardado en el almacenamiento local del nave- gador	47
5.33. Exportar diagrama	48
5.34. Importar diagrama	49
6.1. Aplicación web Draw.io	54
6.2. Aplicación Excalidraw	55

Índice de tablas

5.1. Comparativa del Proyecto plantilla frente a Draw-E-R-App . . .	20
6.1. Comparativa del proyecto frente a alternativas	55

1. Introducción

En desarrollo de software el correcto modelado de la información con la que se trata es fundamental. Una de las técnicas más utilizadas para representar la estructura de un modelo de datos, previo a su implementación en base de datos, es el modelo de diagramas entidad-relación (E-R) [5]. Estos diagramas permiten visualizar las relaciones entre diferentes entidades, así como sus atributos, facilitando así la comprensión y diseño de sistemas de información complejos.

En la era actual, muy centrada en el entorno web surge la necesidad de tener una aplicación web que nos permita, de manera sencilla y accesible en cualquier dispositivo, modelar diagramas entidad-relación y exportarlos fácilmente a tablas SQL. Esta herramienta tiene también un propósito académico, sirve para aprender respecto al paso a tablas SQL de un diagrama E-R.

El objetivo de este proyecto es desarrollar una aplicación web que permita modelar diagramas entidad-relación de manera sencilla, utilizando la biblioteca mxGraph en un entorno React. La aplicación no solo permite la creación y edición de estos diagramas, sino que también permite validarlos y exportarlos a un script SQL listo para ser ejecutado.

Las características principales de esta versión de la herramienta incluyen:

- Aplicación web single page application (SPA) usando el framework React.
- Interfaz de usuario simple y accesible que permite a los usuarios diseñar diagramas E-R de manera visual.
- Uso de la biblioteca mxGraph para el modelado de los diagramas.
- Soporte para modelado y relaciones más básicas de E-R como son:

- Interrelaciones binarias.
- Interrelaciones reflexivas.
- Validación de los diagramas para asegurar la consistencia y corrección de los modelos.
- Exportación e importación de los diagramas validados a un archivo JSON, permitiendo su recuperación para posteriores ediciones.
- Exportación de los diagramas validados a scripts SQL, facilitando así su implementación en bases de datos reales.

La creación de esta aplicación responde a la necesidad de modernizar y simplificar el proceso de modelado de bases de datos, haciendo que sea accesible para un público más amplio y sin requerir de aplicaciones específicas bastando para ello tan solo un navegador web.

En este documento se detalla el desarrollo de la aplicación, desde su concepción y diseño hasta su implementación y validación.

1.1. Estructura de la memoria

- **Introducción:** Descripción de la situación y el tema sobre el que va a tratar el proyecto. Estructura de la memoria y de los anexos.
- **Objetivos del proyecto:** Explicación de los temas que se persiguen en este proyecto.
- **Conceptos teóricos:** Introducción a determinados conceptos necesarios para la comprensión del proyecto.
- **Técnicas y herramientas:** Presentación de metodologías y herramientas que han sido utilizadas para llevar a cabo el proyecto.
- **Aspectos relevantes del desarrollo del proyecto:** Aspectos a destacar durante el desarrollo del proyecto
- **Trabajos relacionados:** Pequeña introducción a proyectos similares y comparativa.
- **Conclusiones y líneas de trabajo futuras:** Conclusiones obtenidas al final del proyecto y posibles ideas futuras.

1.2. Estructura de los anexos

- **Plan de proyecto software:** Planificación temporal y viabilidad económica y legal.

- **Especificación de requisitos:** Objetivos y requisitos a completar durante el desarrollo del proyecto.
- **Especificación de diseño:** Recoge los diseños de datos, procedimental, arquitectónico y de interfaces.
- **Documentación técnica de programación:** Guía técnica del proyecto con conceptos como instalación, organización de carpetas, ejecución del proyecto y pruebas.
- **Documentación de usuario:** Guía destinada al usuario final con explicación paso a paso referente al uso de la aplicación.
- **Anexo de sostenibilización curricular:** Reflexión personal sobre los aspectos de sostenibilidad que se abordan en el trabajo.

1.3. Enlaces

- Página web del repositorio en GitHub [18]
- Despliegue de la aplicación en Vercel [19]

2. Objetivos del proyecto

En este apartado se detallan los objetivos a completar en el desarrollo del proyecto. Distinguiremos los marcados por los requisitos software y los objetivos de carácter técnico necesarios para completar el desarrollo del proyecto.

2.1. Objetivos generales

- Desarrollar una aplicación web para modelar diagramas básicos entidad-relación.
- Proporcionar una interfaz sencilla y fácil de usar para el modelado de diagramas.
- Implementar funcionalidades para añadir y manipular entidades y relaciones.
- Permitir la validación de los diagramas modelados.
- Permitir exportar e importar los diagramas modelados como JSON.
- Exportar los diagramas validados a scripts SQL.

2.2. Objetivos técnicos

- Utilizar React para el desarrollo de la aplicación web.
- Integrar la librería mxGraph para la creación y manipulación de diagramas.
- Implementar un sistema de menús contextuales para facilitar la manipulación de elementos del diagrama.

- Almacenar los datos de los diagramas en una estructura interna eficiente con las siguientes características:
 - Es fácilmente procesable para la generación de las tablas SQL correspondientes.
 - Es fácilmente extensible para expandir las interrelaciones que pueden modelarse.
- Implementar la funcionalidad para validación de diagramas entidad-relación.
- Desarrollar la funcionalidad de exportación de diagramas a scripts SQL.
- Integrar pruebas unitarias y pruebas end-to-end.
- Configurar un flujo CI/CD que ejecute las pruebas y realice un despliegue automático de la aplicación en la plataforma Vercel.
- Utilizar git como sistema de control de versiones, usando como repositorio remoto la plataforma GitHub.

3. Conceptos teóricos

A continuación se detallan algunos de los conceptos más relevantes para la correcta comprensión de este proyecto.

3.1. Aplicación web

Una aplicación web es un programa software que se ejecuta en un servidor web y es accesible a través de un navegador mediante Internet. A diferencia de una página web tradicional, que generalmente se compone de documentos estáticos y contenido multimedia, una aplicación web ofrece interactividad avanzada y para ello requiere el uso combinado de HTML y Javascript.

Las aplicaciones web modernas suelen estar compuestas por una interfaz de usuario (frontend) desarrollada con tecnologías como HTML, CSS y JavaScript [34]. En concreto se suelen usar frameworks como React [9], Vue [2] o Svelte [1] entre otros, siendo React el más popular de todos ellos.

Single Page Application (SPA)

Una Single Page Application (SPA) [27] es un tipo de aplicación web que interactúa con el usuario dinámicamente y se carga en una sola página HTML. En lugar de cargar páginas enteras nuevas desde el servidor, una SPA carga inicialmente la estructura básica de la aplicación y luego actualiza dinámicamente el contenido de la página modificando el DOM o lo que es lo mismo la estructura del documento HTML.

3.2. Desarrollo web moderno

El desarrollo web moderno abarca la creación y mantenimiento de sitios y aplicaciones web. No se trata simplemente de elementos estáticos como HTML y CSS, sino que involucra el uso de JavaScript para agregar interactividad y dinamismo a las aplicaciones. JavaScript es el eje de todo y su mayor complicación es lo mucho que puede variar según el entorno de ejecución [26].

Para gestionar estas diferencias y facilitar el desarrollo, se utilizan herramientas que realizan la transpilación y el empaquetado del código.

Empaquetador

Un empaquetador [24] es una herramienta que procesa y agrupa múltiples archivos de código fuente (JavaScript, CSS, imágenes, etc.) en un solo archivo o en unos pocos archivos optimizados para su posterior ejecución en el navegador.

Transpilador

Un transpilador [29] convierte código escrito en un lenguaje de programación a otro. En el contexto del desarrollo web, un transpilador permite a los desarrolladores escribir código JavaScript con las características más recientes del lenguaje y transformarlo a una versión compatible con la mayoría de los navegadores web actuales. La transpilación puede incluir la conversión de sintaxis moderna de JavaScript (ES6+), JSX (utilizado en React), y otras extensiones a JavaScript estándar que todos los navegadores pueden entender.

Frameworks Frontend

Los frameworks de frontend, como React [9], Vue [2] y Svelte [1], son esenciales en el desarrollo web moderno. Estos frameworks proporcionan estructuras y componentes reutilizables que facilitan la creación de interfaces de usuario dinámicas y complejas. Ayudan a gestionar el estado de la aplicación y la manipulación del DOM [25]. Se basan en un lenguaje cuya base es Javascript pero que puede tener al mismo tiempo secciones con código Javascript, HTML y CSS.

3.3. UI y UX

UI (User Interface) y UX (User Experience) son dos conceptos clave [10] en el desarrollo de aplicaciones web.

UI (User Interface)

La interfaz de usuario comprende tanto el diseño gráfico como los elementos interactivos que se utilizan para interactuar con una aplicación. Una buena UI debe ser intuitiva, atractiva y eficiente.

UX (User Experience)

La experiencia del usuario abarca todos los aspectos de la interacción del usuario con la aplicación. Un buen diseño de experiencia de usuario se enfoca en mejorar la eficiencia y la satisfacción del usuario al interactuar con la aplicación.

3.4. Linter

Un linter [32] es una herramienta que analiza el código fuente para detectar errores de programación, fallos de estilo, y problemas potenciales. Utilizar un linter ayuda a mantener un código limpio, coherente y a revelar potenciales problemas que no se detectan como simples errores sintácticos. Por su naturaleza de extender al compilador tienen cierto componente de opinión y por tanto son configurables en su estilo.

3.5. Integración continua

La integración continua (CI) [16] es una práctica en desarrollo de software que consiste en realizar integraciones automáticas con una gran frecuencia para así poder detectar fallos. Este concepto de integración comprende la compilación (en este caso despliegue) y ejecución de las diferentes pruebas. Herramientas como GitHub Actions facilitan la implementación de CI al automatizar estas tareas.

3.6. Despliegue continuo

El despliegue continuo (CD) [17] extiende la integración continua al proceso de despliegue, automatizando la entrega de cambios en el código a entornos de producción. Esto asegura que el software esté siempre en un estado desplegable y que las nuevas versiones se puedan lanzar rápidamente, mejorando la eficiencia y reduciendo el tiempo y esfuerzo de entrega.

3.7. Validación y pruebas

El proceso de validación y pruebas es fundamental en el desarrollo de software y garantiza que el producto funcione como se espera. Existen varios tipos de pruebas:

Pruebas unitarias

Las pruebas unitarias [35] verifican el funcionamiento de componentes individuales del código, de manera aislada e individual.

Pruebas end-to-end

Las pruebas end-to-end (E2E) [11] simulan el comportamiento del usuario final y prueban la aplicación en su totalidad, desde el punto de vista del usuario final. Estas pruebas aseguran que todos los componentes del sistema funcionen juntos como se espera.

4. Técnicas y herramientas

4.1. mxGraph

mxGraph [21] es una librería JavaScript que permite la creación y manipulación de diagramas interactivos en el navegador y usa elementos vectoriales (SVG) y HTML para renderizar sus elementos. Se utiliza como base para crear aplicaciones que requieran la creación y edición de diagramas. mx-Graph permite a los desarrolladores definir nodos, aristas y otros elementos gráficos, así como gestionar la disposición y estilo de estos elementos.

4.2. React

React [9] es una biblioteca de JavaScript para construir aplicaciones web interactivas. Se basa en componentes reutilizables que gestionan su propio estado. React funciona modificando el HTML que carga el navegador utilizando un DOM virtual para minimizar las actualizaciones en el DOM real, mejorando así el rendimiento de las aplicaciones web.

4.3. Material UI

Material UI [28] es una biblioteca de componentes de interfaz de usuario para React, basada en el diseño de Material Design de Google.

4.4. Biome

Biome [4] es una herramienta que incorpora tanto formateo como linting para JavaScript. Ayuda a mantener un código limpio, coherente y libre

de errores. En este proyecto, Biome se utiliza para asegurar la calidad del código antes de cada commit.

4.5. Herramientas utilizadas para validación y pruebas

Pruebas unitarias

Inicialmente se utilizó Jest [20] para las pruebas unitarias, pero debido a problemas con la transpilación del código, se migró a Vitest [33]. Vitest ofrece una API igual a Jest, con lo que la migración es directa.

Pruebas end-to-end

Para las pruebas funcionales, se utiliza Playwright [30], una herramienta moderna que permite simular las interacciones del usuario con la aplicación en varios navegadores. Playwright ejecuta la aplicación en un entorno real y prueba las funcionalidades desde la perspectiva del usuario final.

Test Driven Development (TDD)

El desarrollo guiado por pruebas de software, o TDD [36] es una práctica de ingeniería de software que requiere el escribir las pruebas primero y refactorizar después. En primer lugar, se escribe una prueba y se comprueba que esta prueba falla. A continuación, se implementa el código que hace que la prueba pase satisfactoriamente y finalmente se refactoriza el código con la seguridad de que este cambio no afectará a la funcionalidad. Esta técnica nos permite pensar en inputs y outputs esperados ayudándonos a desarrollar la funcionalidad desde el test, así mismo nos permite asegurarnos que posteriores cambios no romperán la funcionalidad.

4.6. Herramientas utilizadas para integración continua

La integración continua es una práctica en desarrollo de software que consiste en realizar integraciones automáticas con una gran frecuencia para así poder detectar fallos. Este concepto de integración comprende la compilación (en este caso despliegue) y ejecución de las diferentes pruebas.

Local

Para la integración continua local, se utiliza Lefthook [7], que es una herramienta de hooks que se ejecuta antes de cada commit. Lefthook ejecuta Biome para formatear el código y verificar su calidad en los archivos que están preparados para hacer commit.

GitHub Actions

GitHub Actions [15] es una parte de la plataforma GitHub que permite automatizar flujos de integración continua mediante su definición en archivos YAML. En este proyecto, se utilizan varias acciones:

- Biome: Verifica la calidad del código.
- Tests:
 - Instala dependencias.
 - Instala navegadores de Playwright.
 - Ejecuta tests unitarios.
 - Ejecuta los tests end to end de Playwright con los diferentes navegadores.

4.7. Despliegue continuo

El despliegue continuo se realiza mediante Vercel, que permite mantener versiones de producción y preview. Vercel facilita el despliegue continuo, asegurando que la aplicación esté siempre actualizada y lista para ser utilizada. Permite tener la rama principal constantemente desplegada e integrar los últimos cambios de manera automática, así como probar en un despliegue real los últimos cambios previos a su integración en la rama principal.

4.8. Git

Git [12] es el sistema de control de versiones por excelencia en el desarrollo de software. Es fundamental para trabajar en pequeños y grandes proyectos, sobre todo en aquellos donde se requiera trabajar simultáneamente en equipo.

Git worktrees

Para este proyecto se ha hecho uso extensivo de los git worktrees [13], una característica un tanto desconocida de git que permite hacer un checkout

de un determinado commit o rama y tratarlo como su propio directorio. Permite por tanto cambiar fácilmente entre ramas sin tener problemas con archivos no commiteados.

4.9. GitHub

GitHub es una plataforma online [\[14\]](#) de desarrollo colaborativo que permite usarla como repositorio remoto en un repositorio git. No solo ofrece hosting sino que también nos proporciona diversas utilidades como gestión de issues, revisión de código, pull request y otras muchas opciones.

5. Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Incluye desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación.

Razonamiento tras la elección de este proyecto

Se escogió este proyecto debido a que profesionalmente trabajo muy orientado al desarrollo web, con lo que sería un buen proyecto que añadir a mi portafolio personal. Adicionalmente, siempre me han fascinado aplicaciones como draw.io así que la idea de crear algo similar resultó bastante interesante. En su momento durante la carrera me hubiera gustado contar con una herramienta similar a esta, capaz de generar scripts SQL a partir de un diagrama por lo que este proyecto tenía muchos ingredientes para considerar que podría ser mi trabajo de fin de grado.

5.1. Metodologías

Desarrollo ágil

Para el desarrollo de este proyecto se ha usado la metodología de desarrollo ágil. No se ha aplicado estrictamente pero sí en bastante grado. Se ha realizado un desarrollo iterativo agrupando las tareas en sprints de aproximadamente 3-4 semanas cada uno de ellos. Con cada sprint se definían una serie de tareas cuyo fin era obtener un producto final que cumpliera los

objetivos marcados inicialmente. Al realizarse este desarrollo incremental permitía centrarse en estas tareas individuales, organizándose de una mejor manera. Se ha combinado esta división en sprints con los issues de GitHub y los milestones para representar los sprints.

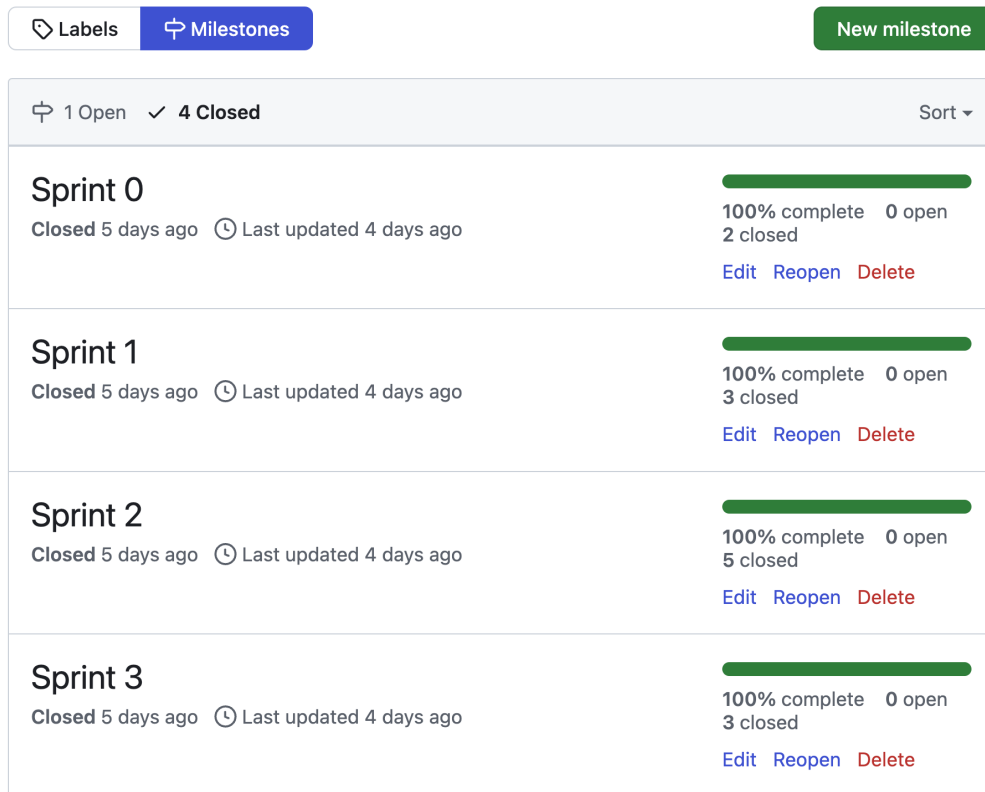


Figura 5.1: Milestones en GitHub

Cada issue se etiquetó con labels correspondientes a qué tipo de cambio representaba.

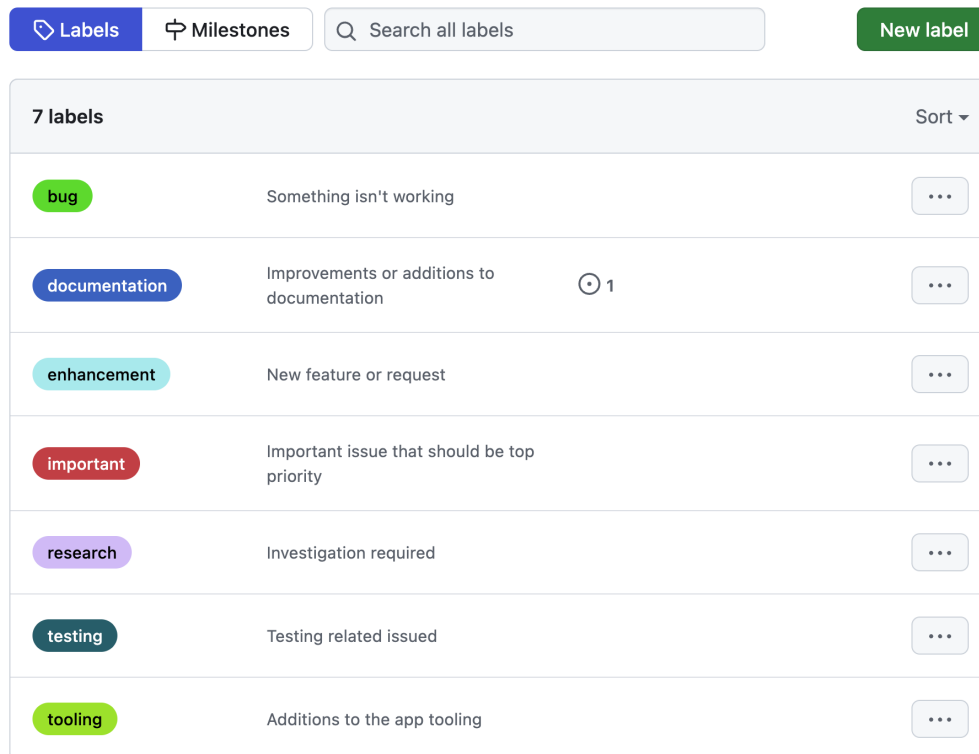


Figura 5.2: Labels en GitHub

- Bug: Issue que representa errores en la aplicación.
- Documentation: Issue que representa cambios en la documentación de la aplicación.
- Enhancement: Issue que representa una nueva característica a integrar en la aplicación.
- Important: Label para asignar una categoría de importancia superior a un issue.
- Research: Issue para investigar un determinado área o concepto.
- Testing: Issue que introduce cambios en los tests de la aplicación.
- Tooling: Issue que introduce cambios en determinadas herramientas de desarrollo de la aplicación, como el linter.

Flujo git

El uso de git ha sido de una manera determinada, basada en Gitflow [3]. No se ha aplicado estrictamente en todos los cambios. Pero sí se ha hecho con las tareas definidas en cada sprint, sobre todo aquellas que integraban

funcionalidades. Este enfoque, combinado con el despliegue continuo en Vercel, proporcionaba un despliegue "Preview" desde el que evaluar la calidad de estos cambios a integrar. No solo eso, también permitía compartir el enlace con el tutor y recibir su feedback de un despliegue real.

git worktrees

Git worktrees [13] es una herramienta que nos proporciona git que es muy poderosa pero que curiosamente muy desconocida. Al utilizar worktrees, se puede cambiar de una rama a otra simplemente cambiando de directorio, puesto que cada worktree representa una rama y es un directorio. No hay necesidad de hacer stash de cambios no comiteados¹. Esto ha permitido una transición fluida entre diferentes ramas de trabajo y ha facilitado la integración de correcciones en la rama principal (main).

El uso de git worktrees junto con Gitflow ha ayudado enormemente a moverse entre ramas de funcionalidades, revisar código y asegurar que los cambios que se introducían fueran correctos. Es una funcionalidad de git que muchos desarrolladores debieran conocer y probar.

Test driven development

El desarrollo guiado por pruebas (Test Driven Development, TDD) [36] ha sido clave en la implementación de ciertas funcionalidades importantes de la aplicación, como son la validación de los diagramas entidad-relación y la generación de scripts SQL. Inicialmente se definían los requisitos de estas funciones, previo paso a su implementación. De esta manera en un primer momento ya se decidía que debía devolver la función desde unos inputs conocidos, lo cuál no ayudaba solo a asegurarse de que la función fuera correcta sino que facilitaba su implementación.

¹Un stash es una operación en git que permite guardar temporalmente los cambios no comiteados para poder cambiar de rama sin perder esos cambios.


```
✓ tests/unit/validation.test.js (10)
  ✓ General validation function (1)
    ✓ correct graph return true
  ✓ Non repeated entity or n:m relation name (2)
    ✓ entities can't have repeated names
    ✓ N:M relations and entities can't have repeated names
  ✓ Non repeated attributes in entities or n:m relations (2)
    ✓ entities can't have repeated attributes names
    ✓ N:M relations can't have repeated attributes names
  ✓ Every entity should have at least one attribute (2)
    ✓ entities must have at least one attribute
    ✓ N:M relations must have at least one attribute
  ✓ Relations (3)
    ✓ Every relation should connect two entities (can be the same at both sides)
    ✓ Cant be relations with attributes if they are not N:M
    ✓ Every relation should have valid cardinalities

Test Files  1 passed (1)
Tests       10 passed (10)
Start at    13:17:57
Duration    27ms
```

Figura 5.3: Ejecución automática de tests unitarios

5.2. Desarrollo del proyecto

Primeros pasos en el desarrollo del proyecto

Durante las primeras semanas de desarrollo se decidió qué tecnologías usar. Inicialmente existía la duda se si hacerlo con Vanilla Javascript o usar un framework frontend como React. Finalmente se optó por usar React debido a las facilidades que puede dar a la hora de crear una aplicación interactiva y con estado dinámico. También es clave en esta decisión comentar que personalmente es una tecnología que uso profesionalmente y que por tanto no iba a requerir formación más allá de la propia integración con mxGraph. En una primera fase, se intentó utilizar la librería maxGraph [22] (sucesor de mxGraph) junto con TypeScript [23], lo que nos habría permitido contar con la seguridad de un tipado estático. Sin embargo, hubo problemas que hicieron desestimar esta opción: maxGraph es una librería que surgió con la idea de continuar el desarrollo de mxGraph que se encuentra en su fin de vida, sin embargo se han hecho bastantes cambios en la API con respecto

a mxGraph y la documentación no se ha completado. Esto implica que para encontrar los métodos para la nueva API de maxGraph equivalentes a la antigua de mxGraph había que indagar constantemente en el código fuente en busca de las nuevas APIs.

Punto de partida

Finalmente se acabó usando como punto de partida a un proyecto plantilla ² que integraba mxGraph con React. Esta plantilla proporcionaba una toolbar y permitía inicializarla con elementos que podían ser arrastrados y soltarlos al canvas donde se añadían visualmente. Este punto será la base desde la que partió el desarrollo.

Este proyecto plantilla cuenta con la inicialización de mxGraph así como una estructura de archivos para crear una barra lateral de herramientas. No obstante, tan solo nos sirve como base.

Apreciamos las siguientes diferencias entre el proyecto base y la solución final de este trabajo de fin de grado.

Características	Proyecto plantilla	Draw-E-R-App
Añadir rectángulos	X	-
Relacionar objetos directamente	X	-
Cambiar nombres de los objetos	X	X
Modelar diagramas E-R	-	X
Añadir entidades	-	X
Añadir atributos a entidades	-	X
Añadir relaciones	-	X
Configurar relaciones	-	X
Validar un diagrama	-	X
Generar un script SQL	-	X
Exportar el diagrama como JSON	-	X
Importar y recrear el diagrama	-	X
Recuperar el estado entre sesiones	-	X

Tabla 5.1: Comparativa del Proyecto plantilla frente a Draw-E-R-App

²<https://codesandbox.io/p/sandbox/react-kos1g?file=/src/App.js>

```

React.useEffect(() => {
  // Create the graph if it's not loaded when the component
  renders
  if (!graph) {
    mxEvent.disableContextMenu(containerRef.current);
    setGraph(new mxGraph(containerRef.current));
  }
  // Wait for the graph to be loaded
  if (graph) {
    setInitialConfiguration(graph, toolbarRef);

    // Updates the display
    graph.getModel().endUpdate();
    graph.getModel().addListener(mxEvent.CHANGE, onChange);
    graph.getSelectionModel().addListener(mxEvent.CHANGE,
      onSelect);
    graph.getModel().addListener(mxEvent.ADD, onElementAdd);
    ;
    graph.getModel().addListener(mxEvent.MOVE_END,
      onDragEnd);
  }
}, [graph]);

```

Figura 5.4: Inicialización de mxGraph

```

return (
  <div className="mxgraph-container">
    <div className="mxgraph-toolbar-container">
      <div className="mxgraph-toolbar-container" ref={
        toolbarRef} />
    </div>
    <div ref={containerRef} className="mxgraph-drawing-
      container" />
  </div>
);

```

Figura 5.5: HTML renderizado por el componente de React **DiagramEditor.js**

Implementando las primeras funcionalidades

Añadir entidades

Con esta base sobre la que trabajar se implementó la funcionalidad inicial para poder añadir entidades al canvas arrastrando desde la toolbar. A estas entidades se les puede cambiar el nombre haciendo doble click en ellas y en este punto y debido al comportamiento por defecto de la librería se pueden relacionar directamente entre sí. Esto supone un problema que habrá que resolver posteriormente, en nuestro caso concreto queremos que las entidades se relacionen por medio de elementos Relación y no directamente.

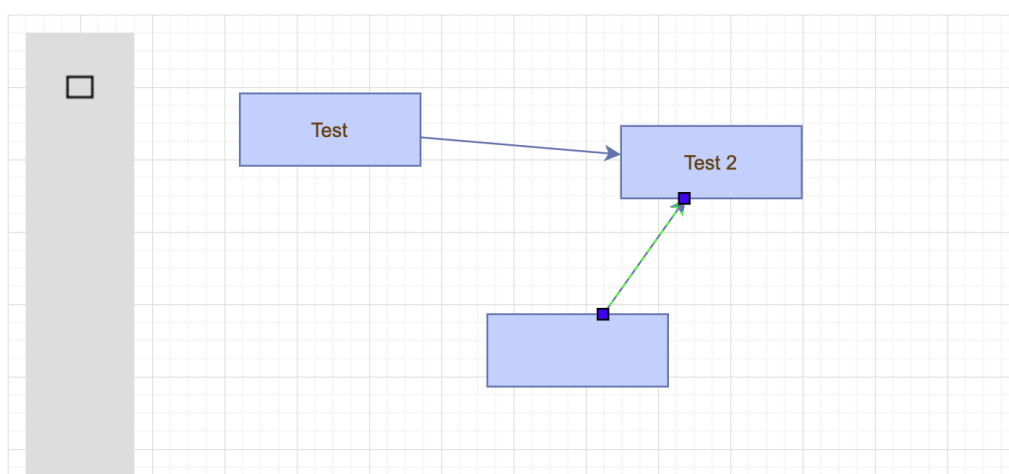


Figura 5.6: Aplicación tras implementar la posibilidad de añadir entidades

Representación interna del diagrama E-R

Se comienza también a trabajar en la funcionalidad que nos permita pasar de las celdas con las que trabaja mxGraph, que representan los distintos elementos que se han creado en el canvas, a una representación interna más acorde a lo que es un diagrama entidad relación. Hay que conocer qué elemento es una entidad, cuáles son sus atributos, qué elementos está conectando una relación, etc. Al estar trabajando con Javascript la preferencia es usar un objeto por su facilidad a ser convertido a JSON.

Aunque esta captura es de un punto más avanzado donde ya se pueden añadir atributos a las entidades, ya representa la información que se quiere guardar en esta representación interna.

```
{
  "entities": [
    {
      "idMx": "2",
      "name": "Entidad",
      "position": {
        "x": 254,
        "y": 130
      },
      "attributes": [
        {
          "idMx": "3",
          "name": "Atributo",
          "position": {
            "x": 374,
            "y": 130
          }
        }
      ]
    }
  ],
  "relations": []
}
```

Figura 5.7: Primera versión de la representación interna del diagrama E-R

- **idMx:** Es el id que tiene este elemento en las celdas de la librería `mxGraph`.
- **name:** Es el nombre que tiene esta entidad.
- **position:** Coordenadas de esta entidad en el canvas.
- **attributes:** Array de atributos, de cada uno de ellos se guardarán sus datos.

Este paso es realmente importante puesto que hay que establecer condiciones y límites que solo con `mxGraph` no podremos. Será clave en todo el desarrollo posterior, , puesto que, por ejemplo: si un atributo es clave habrá que guardarlo como tal, si una relación es N:M podrá tener atributos, etc. La validación del grafo tratará directamente con esta representación.

Menús contextuales según el elemento seleccionado

Otra funcionalidad importante especialmente en el desarrollo de la interfaz de usuario es el de que la toolbar muestre diferentes botones según el

elemento seleccionado. Con React se crea un estado que evalúa qué elemento está seleccionado en cada momento y cada botón que se crea en la toolbar comprueba el estilo del elemento seleccionado para saber si es una entidad o atributo y se muestra o no.

```
const renderAddAttribute = () => {  
  if (selected?.style?.includes("shape=rectangle")) {  
    return (  
      <button  
        type="button"  
        className="button-toolbar-action"  
        onClick={addAttribute}  
      >  
        Anadir atributo  
      </button>  
    );  
  }  
};
```

Figura 5.8: Botón contextual para entidades

Añadiendo atributos a las entidades

La siguiente funcionalidad a implementar fue la de añadir atributos a las entidades, para ello hizo falta añadir un botón contextual al seleccionar una entidad y desde ahí en su versión inicial nos dejaba escoger entre si se quería añadir un atributo primario (clave) o uno normal. Esta funcionalidad fue relativamente problemática puesto que el atributo no consta solo de una elipse sino que tiene 3 partes diferenciadas que han de agruparse: el edge, la elipse y la label que contiene el nombre del atributo. Además la etiqueta ha de emplazarse a la derecha de la elipse y mantenerse en esa posición.

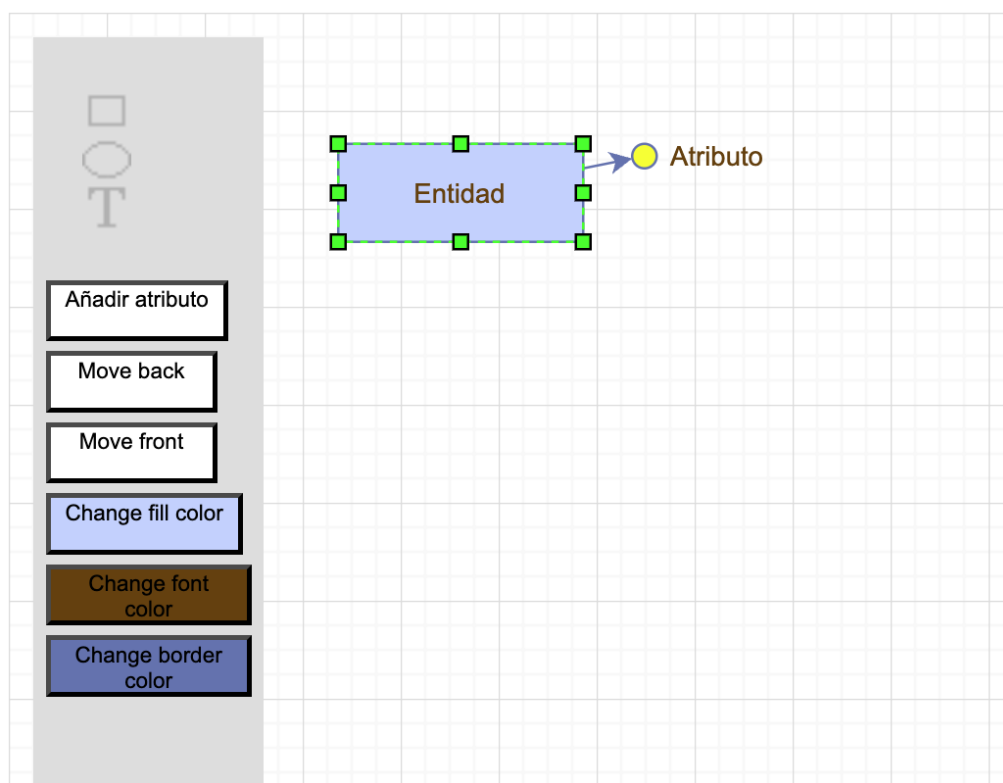


Figura 5.9: Añadir atributos a una entidad

Creando un flujo de CI/CD

Con la aplicación en un estado temprano pero estable, no iba haber más cambios de librerías o frameworks, se decidió añadir un flujo completo de integración continua que garantizase la corrección del código (y un estilo común en caso de que en el futuro colaboren más desarrolladores) y su correcto funcionamiento. Se usaron algunas de las siguientes utilidades.

Biome y Lefthook

Se añadió Biome como formateador y como linter. Biome ejecuta dos funciones que, tradicionalmente, en el desarrollo web se encuentran desacopladas en diferentes herramientas. Funciona de tal modo que formatea el código de manera automática y comprueba posibles errores (que no tienen por qué ser sintácticos), resolviendo aquellos que puede de forma automática y notificando, con una serie de diagnósticos, aquellos que no.

```

$ npm run lint
> react@1.0.0 lint
> npx @biomejs/biome lint --apply ./src

./src/utils/sql.js:178:11 parse
-----
  * Const declarations must have an initialized value.

176 |     } // Case where one side has (0,1) cardinality or both sides have equal m
inimum cardinality
177 |     let tableWithForeignKey;
> 178 |     const tableWithoutForeignKey;
      |           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
179 |     let foreignKeySide;
180 |     let primaryKeySide;

i This variable needs to be initialized.

```

Figura 5.10: Muestra de una ejecución de Biome con errores

Se añadió también Lefthook al proyecto. Se ejecuta de forma local como un hook pre-commit, esto quiere decir que cuando hacemos un commit Lefthook ejecuta una determinada acción con aquellos ficheros (o hunks) que hayamos añadido para hacer commit. En este caso ejecuta Biome sobre todos ellos.

```

🔧 lefthook v1.6.11  hook: pre-commit

| check ›

Checked 0 files in 1332µs. No fixes needed.

-----

summary: (done in 0.48 seconds)
✓ check
[45-documentation e658410] test
Date: Sun Jun 30 12:30:07 2024 +0200
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 test
create mode 100644 test.js

```

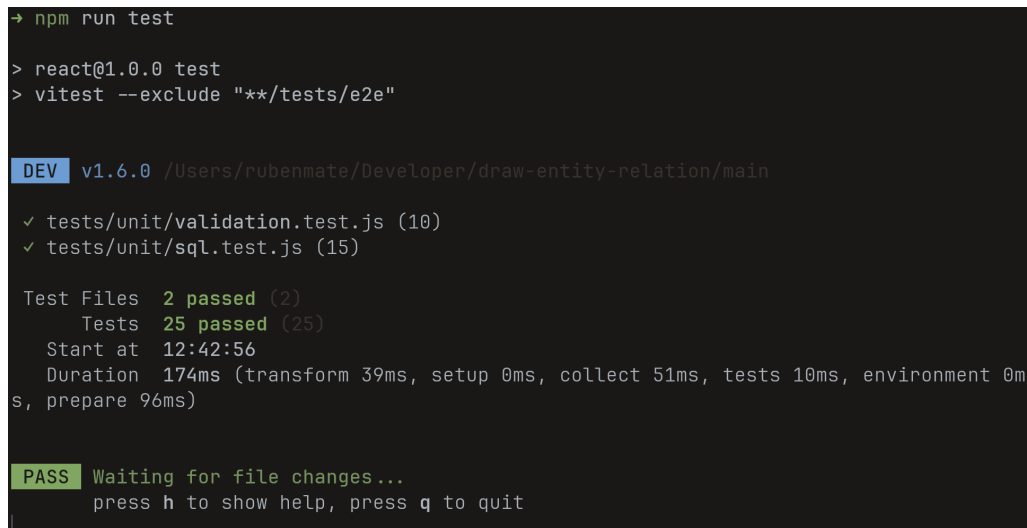
Figura 5.11: Muestra de una comprobación previa a un commit con Lefthook

Testing

El siguiente paso para verificar la calidad y funcionamiento del código era añadir pruebas. Se añadieron dos herramientas aunque inicialmente tan solo se usó la segunda.

- Jest: Framework de testing para Javascript que ejecuta tests unitarios.
- Playwright: Framework de testing que ejecuta tests end to end.

Los tests unitarios inicialmente no se implementaron puesto que la mayor parte de la funcionalidad era a nivel de interfaz. Se utilizarán posteriormente en el desarrollo de las funcionalidades de validación del diagrama y generación del script SQL.



```
→ npm run test
> react@1.0.0 test
> vitest --exclude "**/tests/e2e"

DEV v1.6.0 /Users/rubenmate/Developer/draw-entity-relation/main

✓ tests/unit/validation.test.js (10)
✓ tests/unit/sql.test.js (15)

Test Files 2 passed (2)
Tests 25 passed (25)
Start at 12:42:56
Duration 174ms (transform 39ms, setup 0ms, collect 51ms, tests 10ms, environment 0ms, prepare 96ms)

PASS Waiting for file changes...
press h to show help, press q to quit
```

Figura 5.12: Ejecución de pruebas unitarias

Los tests end to end se ejecutan con Playwright, y consisten en una serie de pruebas que se comportan como el usuario final. Van interactuando con la interfaz pulsando botones y esperando ciertos elementos. En esta figura posterior podemos ver la ejecución de uno de ellos desde su interfaz gráfica, se pueden ver las capturas que va haciendo el framework de tests de la ejecución segundo a segundo.

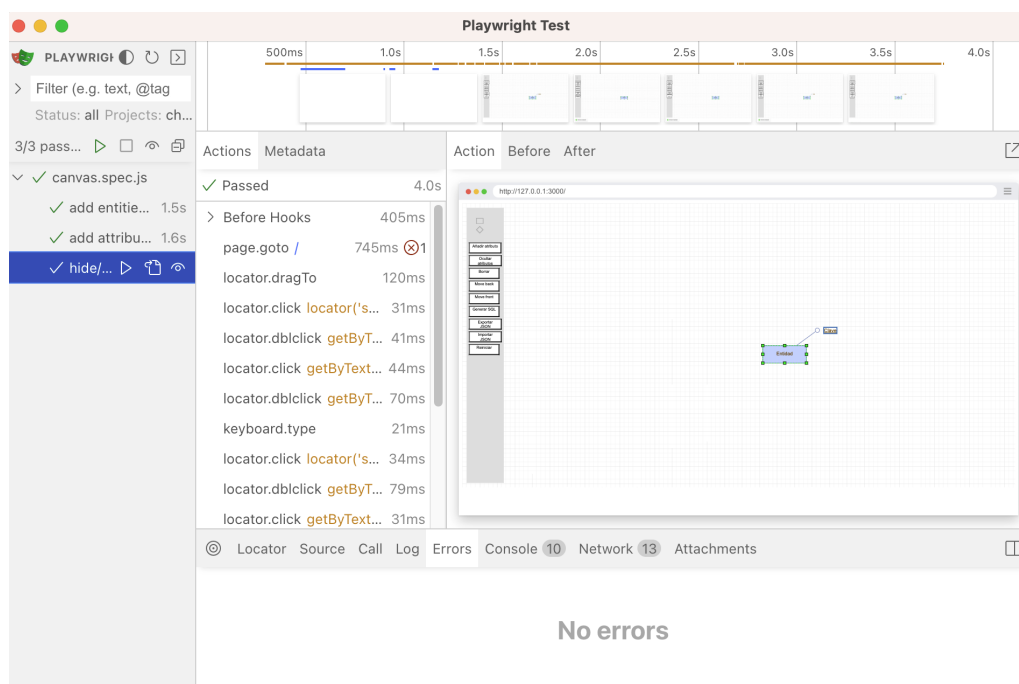


Figura 5.13: Ejecución de pruebas End to End

GitHub Actions

Ninguna de estas herramientas que hemos añadido tienen sentido si no nos acordamos de ejecutarlas. Para ello se ha decidido hacer uso de las GitHub Actions que las ejecutarán automáticamente al subir cambios a la rama principal o a aquellas ramas destinadas a integrarse con la rama principal.

En este proyecto, se utilizan varias acciones:

- Biome: Verifica posibles problemas en el código.
- SonarCloud: Da un diagnóstico de la calidad del código.
- Tests:
 - Instala dependencias.
 - Instala navegadores de Playwright.
 - Ejecuta tests unitarios.
 - Ejecuta los tests end to end de Playwright con los diferentes navegadores.
- Despliegue en Vercel

Inicialmente se había creado una action que instalaba las dependencias del proyecto y hacía la build pero al integrar Vercel se optó por eliminarla puesto que con Vercel ya se está haciendo la instalación y despliegue.

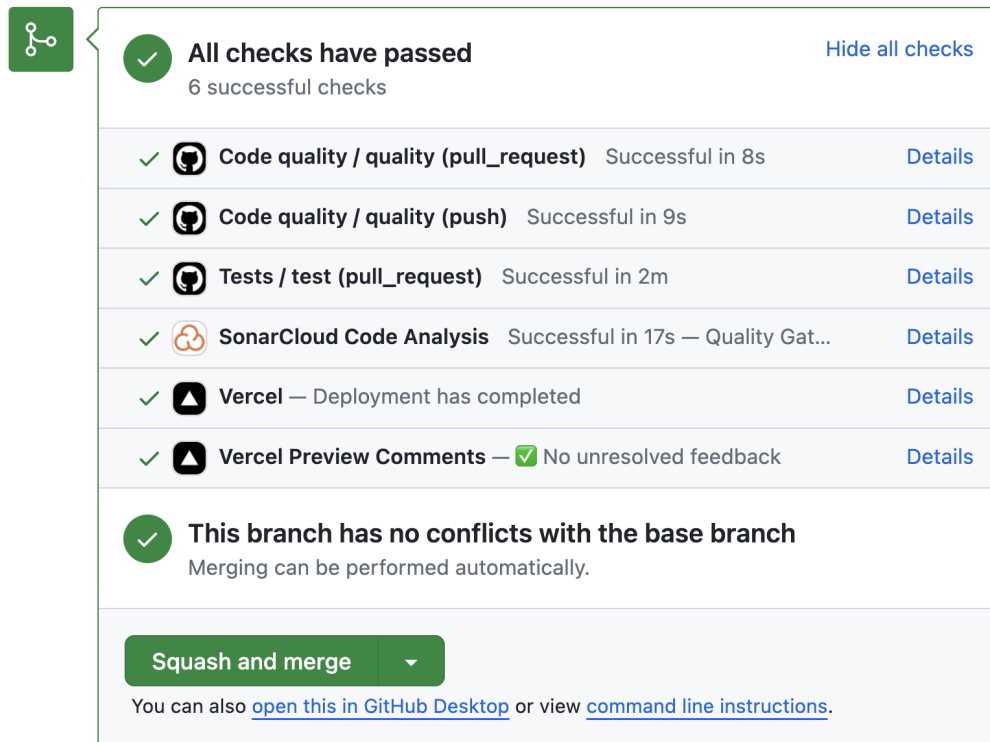


Figura 5.14: Ejecución de las diferentes GitHub Actions en nuestro proyecto

Gestionando atributos clave

Se cambió el modo de añadir atributos clave así como su estética. Los edges dejaron de ser dirigidos y en lugar de colorear la elipse de una forma determinada en su lugar se subraya aquel atributo que sea clave.

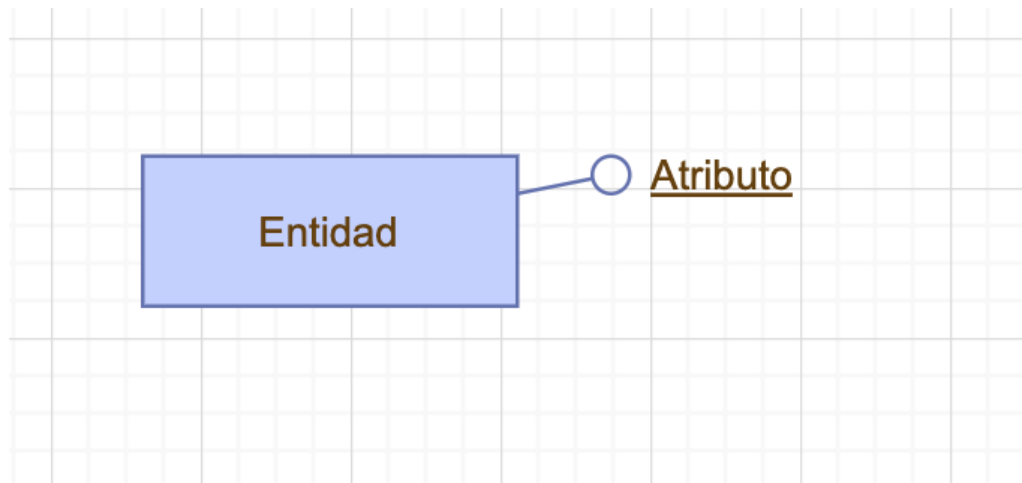


Figura 5.15: Atributo clave subrayado con elipse no coloreada y lado no dirigido

También se elimina la posibilidad de elegir añadir un atributo como clave, en su lugar el primer atributo que se añade a una entidad será clave y el resto no. Esto requiere un botón contextual en atributos no clave para poder convertirlos en clave eliminando esta propiedad de aquel que fuera clave hasta ese momento.

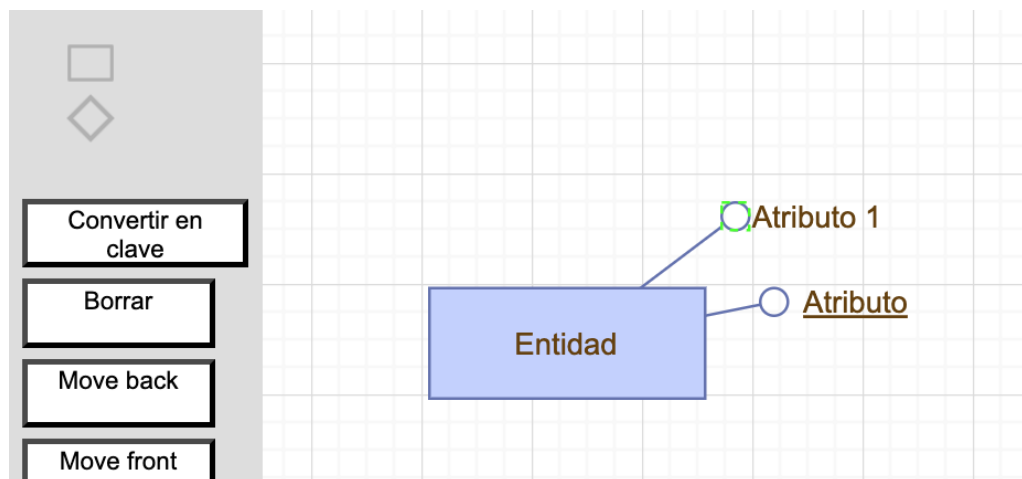


Figura 5.16: Botón contextual para convertir un atributo que no es clave en clave

Implementando relaciones

Con las entidades y sus atributos ya implementados en la aplicación nos faltaba la parte de las relaciones. Es a su vez una de las que tiene mayor complejidad.

Configurando relaciones

Era necesario un modo de añadir relaciones. Se optó por añadir el elemento rombo representando una relación y mostrar botones contextuales con las siguientes acciones.

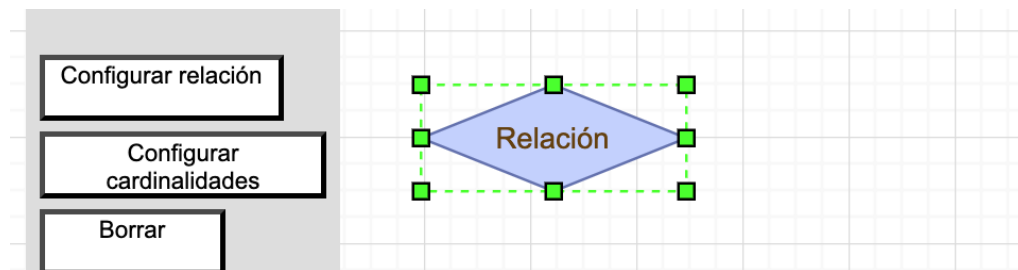
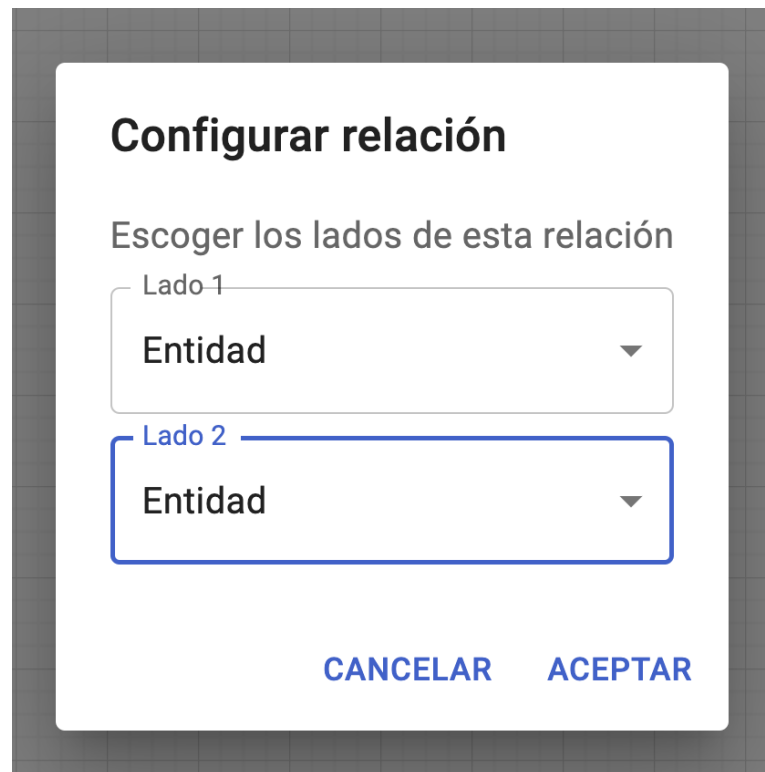


Figura 5.17: Botones contextuales al seleccionar una relación

Al hacer click en configurar relación se nos muestra una interfaz con Material UI con dos selectores para escoger qué entidades va a vincular esta relación.



The image shows a modal dialog box titled "Configurar relación" (Configure relationship). Below the title is the instruction "Escoger los lados de esta relación" (Choose the sides of this relationship). There are two input sections: "Lado 1" (Side 1) and "Lado 2" (Side 2). Each section contains a dropdown menu with the text "Entidad" (Entity) and a downward arrow. The "Lado 2" section is highlighted with a blue border. At the bottom of the dialog are two buttons: "CANCELAR" (Cancel) and "ACEPTAR" (Accept).

Figura 5.18: Interfaz de configuración de los lados de una relación

Tan solo falta el escoger las cardinalidades de la relación, se realizan en el siguiente menú contextual. Sigue la misma dinámica que la propia configuración de las relaciones.

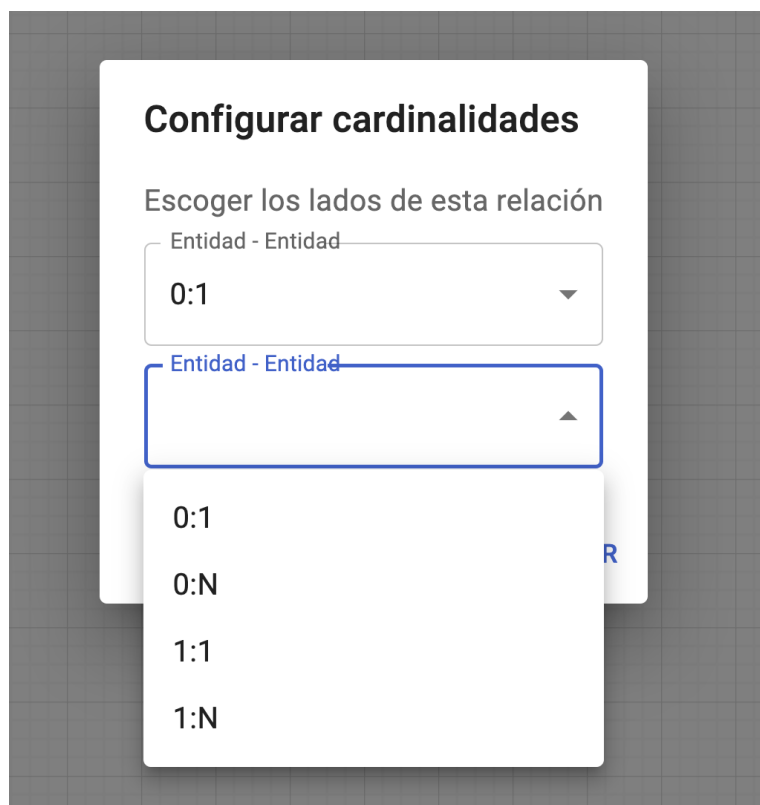


Figura 5.19: Interfaz de configuración de las cardinalidades de los lados de una relación

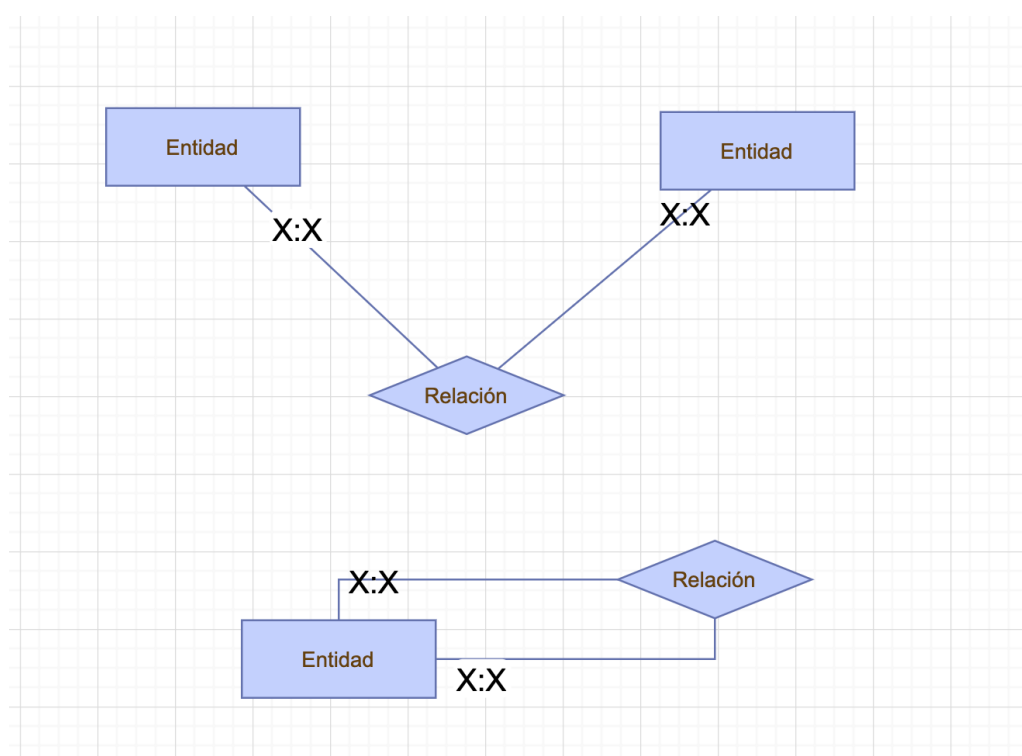


Figura 5.20: Relaciones en el canvas

Todos estos cambios hemos de reflejarlos en nuestra estructura interna en memoria del diagrama donde ahora también estamos guardando las relaciones.

La información **canHoldAttributes** y el campo `attributes` de la relación tendrá relevancia en el siguiente apartado.

Relaciones N:M

Las relaciones que forman una relación varios a varios pueden contener atributos, algo que no estaba implementado. Para implementar esta función se realiza en el paso de configuración de la relación, cuando detectamos que es N:M establecemos en la propia relación una clave `canHoldAttributes: true` y guardamos los atributos como si de una entidad se tratase en un array. Se modifica la interfaz para que al seleccionar una relación N:M nos deje también añadir atributos (con la salvedad de que no puedan ser clave) y ocultarlos.

```
{
  ...
  "relations": [
    {
      "idMx": "8",
      "name": "Relacion",
      "position": {
        "x": 215,
        "y": 110
      },
      "side1": {
        "idMx": "11",
        "cardinality": "0:1",
        "cell": "11",
        "entity": {
          "idMx": "2"
        }
      },
      "side2": {
        "idMx": "12",
        "cardinality": "1:N",
        "cell": "12",
        "entity": {
          "idMx": "5"
        }
      },
      "canHoldAttributes": false,
      "attributes": []
    }
  ]
}
```

Figura 5.21: Diagrama interno guardando relaciones

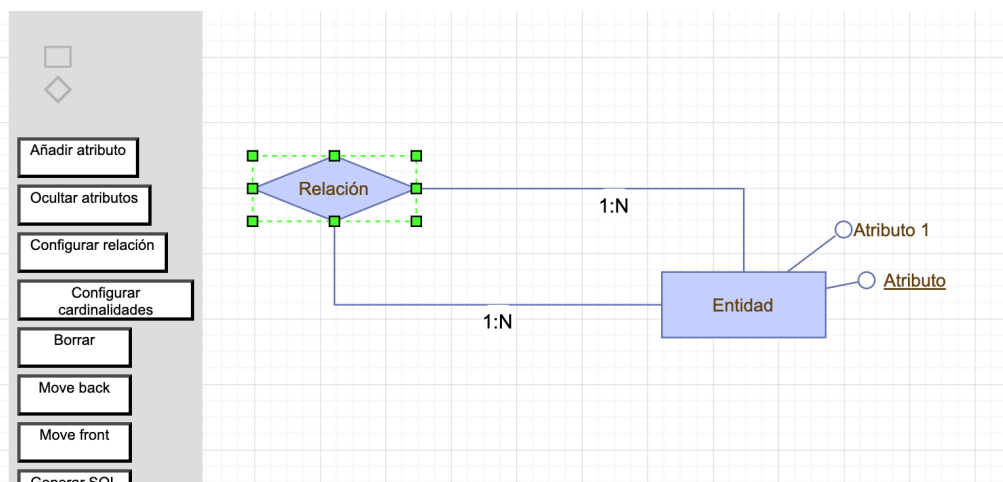


Figura 5.22: Interfaz contextual para una relación N:M

Validación del diagrama y generación SQL

En este punto la parte de modelado está prácticamente finalizada y nos falta tan solo el validar el diagrama y poder exportarlo a un script SQL. Como últimos retoques se implementa una prevención que comprueba si se está usando el placeholder **Relación**, **Entidad** o **Atributo** con el que se insertan los elementos para añadir un número incremental al final (p.e., Relación 1, Relación 2 ...), y evitar que se repitan. También se cambia el framework de Jest a Vitest por problemas en la transpilación de los tests.

Validación del diagrama

Para esta función y la de generación del script SQL se hizo uso intensivo del TDD. Definimos grafos de ejemplo con y sin errores y desde ese punto se implementaron las diversas funciones que han de comprobar la validez del diagrama. Se están comprobando los siguientes criterios:

- Nombres de entidades repetidos (las relaciones N:M cuentan como entidad).
- Nombres de atributos repetidos (en una misma entidad).
- Nombres de relaciones repetidos.
- Entidades sin atributos.
- Entidades sin clave primaria.
- Entidades con más de una clave.
- Atributos clave en relaciones N:M.
- Relaciones sin conectar.
- Relaciones sin lados con cardinalidades válidas. Existe una particularidad en este caso, las relaciones 1:1-1:1. Se controla que no se pueda establecer también a nivel de interfaz (aparte de en la función de validación).³
- Ninguna relación que no sea N:M puede contener atributos.
- El diagrama no puede estar vacío.

³Al escoger 1:1 en alguno de los lados se elimina como opción del otro.

```
export function validateGraph(graph) {
  const diagnostics = {
    noRepeatedNames: true,
    noRepeatedAttrNames: true,
    noEntitiesWithoutAttributes: true,
    noEntitiesWithoutPK: true,
    noUnconnectedRelations: true,
    noNotValidCardinalities: true,
    notEmpty: true,
    isValid: true,
  };

  // The graph is empty
  if (graph.entities.length === 0 && graph.relations.length
    === 0) {
    diagnostics.notEmpty = false;
    diagnostics.isValid = false;
  }

  // Check for repeated entity names
  if (repeatedEntities(graph)) {
    diagnostics.noRepeatedNames = false;
    diagnostics.isValid = false;
  }

  // Check for repeated attribute names in the same entity
  if (repeatedAttributesInEntity(graph)) {
    diagnostics.noRepeatedAttrNames = false;
    diagnostics.isValid = false;
  }

  // Check for entities without attributes
  if (entitiesWithoutAttributes(graph)) {
    diagnostics.noEntitiesWithoutAttributes = false;
    diagnostics.isValid = false;
  }
}
```

Figura 5.23: Función de validación del diagrama con diagnósticos (Parte 1)

```
// Check for entities without a primary key attribute
if (entitiesWithoutPK(graph)) {
    diagnostics.noEntitiesWithoutPK = false;
    diagnostics.isValid = false;
}

// Check for unconnected relations
if (relationsUnconnected(graph)) {
    diagnostics.noUnconnectedRelations = false;
    diagnostics.isValid = false;
}

// Check for relations with invalid cardinalities
if (cardinalitiesNotValid(graph)) {
    diagnostics.noNotValidCardinalities = false;
    diagnostics.isValid = false;
}

return diagnostics;
}
```

Figura 5.24: Función de validación del diagrama con diagnósticos (Parte 2)

Generación del script SQL

La generación del script SQL fue una de las partes más complicadas de implementar. Requiere un paso intermedio donde recorremos el diagrama y cogemos las relaciones y sus correspondientes entidades procesándolas. Si una entidad ya ha sido procesada como parte de una relación la quitamos de las entidades sueltas puesto que no queremos tablas repetidas.

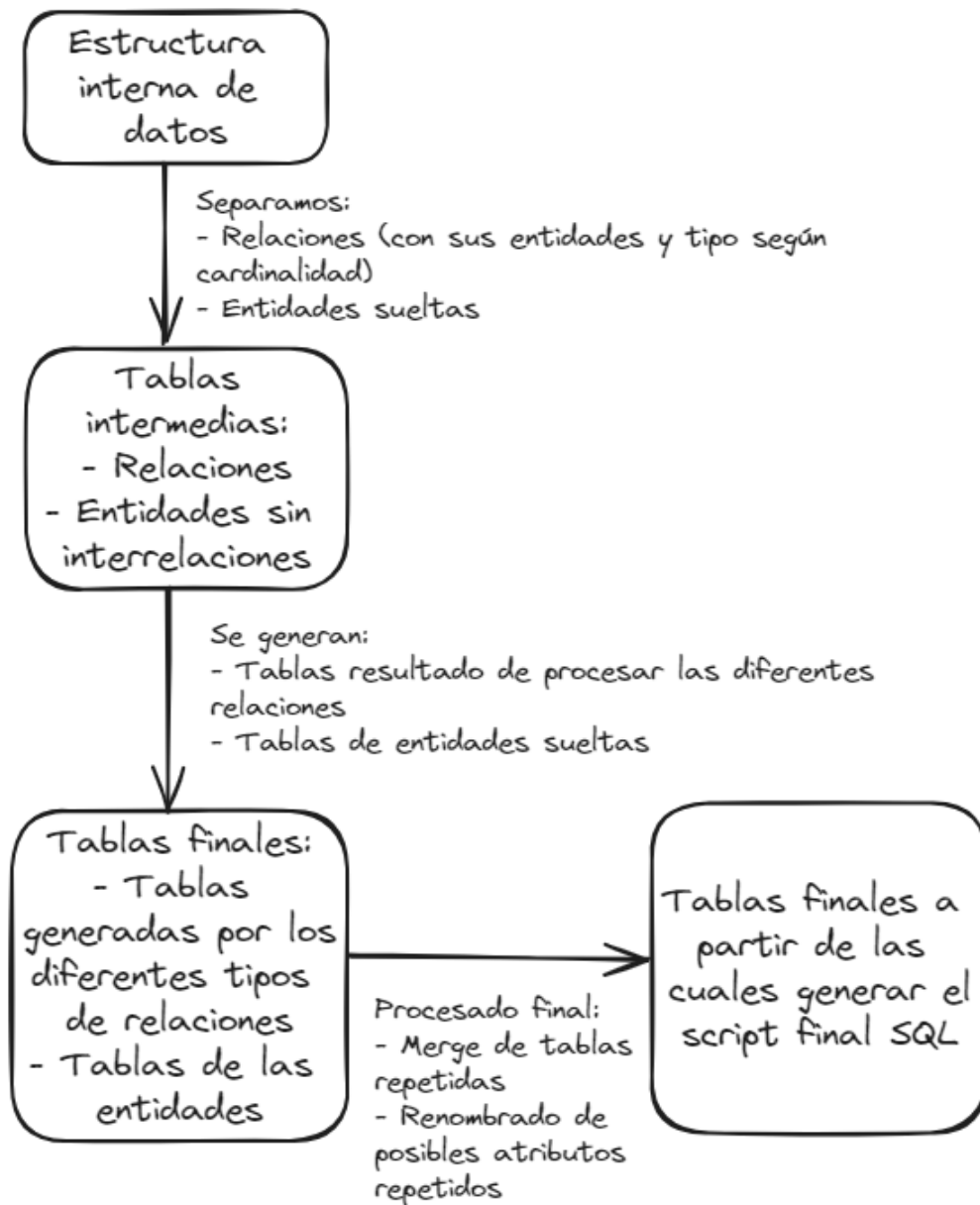


Figura 5.25: Flujo para procesar la estructura interna y generar el script SQL

```
function processRelation(relation) {
  const side1 = relation.side1;
  const side2 = relation.side2;
  const cardinalityType = getCardinalityType(
    side1.cardinality.split(":")[1],
    side2.cardinality.split(":")[1],
  );

  const table = {
    name: relation.name,
    type: cardinalityType,
    side1: {
      entity: entities.find((e) => e.idMx === side1.
        entity.idMx),
      cardinality: {
        minimum: side1.cardinality.split(":")[0],
        maximum: side1.cardinality.split(":")[1],
      },
    },
    side2: {
      entity: entities.find((e) => e.idMx === side2.
        entity.idMx),
      cardinality: {
        minimum: side2.cardinality.split(":")[0],
        maximum: side2.cardinality.split(":")[1],
      },
    },
    attributes: [...relation.attributes],
  };

  // Mark entities as used
  usedEntities.add(side1.entity.idMx);
  usedEntities.add(side2.entity.idMx);

  return table;
}
```

Figura 5.26: Procesado de relaciones (Parte 1)


```
// Process relations first
for (const relation of graph.relations) {
    tables.push(processRelation(relation));
}

// Add remaining entities as tables
for (const entity of entities) {
    if (!usedEntities.has(entity.idMx)) {
        tables.push(entity);
    }
}

return tables;
}
```

Figura 5.27: Procesado de relaciones (Parte 2)

Con este array que contiene relaciones y entidades sueltas se llama a la función que genera las tablas finales haciendo merge en caso de que este procesado genere tablas repetidas. El paso final es convertir este objeto interno en un string SQL.

```

for (const table of tables) {
  let processedTablesArray;
  switch (table.type) {
    case "1:1":
      processedTablesArray = process11Relation(table);
      break;
    case "1:N":
      processedTablesArray = process1NRelation(table);
      break;
    case "N:M":
      processedTablesArray = processNMRelation(table);
      break;
    default:
      processedTablesArray = [table];
      break;
  }

  // Add the processed tables to the map, merging attributes
  // if needed
  for (const processedTable of processedTablesArray) {
    if (tableMap.has(processedTable.name)) {
      const existingTable = tableMap.get(processedTable.name);
      const existingAttributes = new Set(
        existingTable.attributes.map((attr) => attr.name),
      );
      processedTable.attributes.forEach((attr) => {
        if (!existingAttributes.has(attr.name)) {
          existingTable.attributes.push(attr);
        }
      });
    } else {
      tableMap.set(processedTable.name, processedTable);
    }
  }
}

```

Figura 5.28: Procesado de tablas y merge

```

for (const table of tableMap.values()) {
  table.name = removeAccents(table.name);
  // Iterate over tables. Remove the accents, check for
  // repeated attributes with the
  // same name. Change the name of the repeated items so that
  // there is no repetition
  const attributeNames = new Set();
  table.attributes.forEach((attr) => {
    let baseName = removeAccents(attr.name);
    let uniqueName = baseName;
    if (attributeNames.has(uniqueName)) {
      let counter = 1;
      uniqueName = `${baseName}_${counter}`;
      while (attributeNames.has(uniqueName)) {
        counter++;
        uniqueName = `${baseName}_${counter}`;
      }
    }
    attr.name = uniqueName;
    attributeNames.add(uniqueName);
  });
}

```

Figura 5.29: Procesado de tablas finales para eliminar acentos y evitar posibles atributos repetidos.

```

// Generate SQL script from the table map
let sqlScript = "";
let foreignKeyScript = "";
for (const table of tableMap.values()) {
  sqlScript += createTableSQL(table) + "\n\n";
  foreignKeyScript += createForeignKeySQL(table) + "\n";
}

return sqlScript.trim() + "\n\n" + foreignKeyScript.trim();

```

Figura 5.30: Generación del string SQL final

Inicialmente, para las foreign keys, se hacía el 'REFERENCES' en la propia tabla. Esto no siempre es viable, puesto que requiere que la tabla de la que es clave foránea haya sido creada previamente. Se cambió a crear las tablas sin hacer 'REFERENCES' a otras tablas y en el paso final de generación del string SQL incluir tantos 'ALTER TABLE' como sean necesarios para crear esas claves foráneas.

```
ALTER TABLE A ADD CONSTRAINT FK_a1_R1 FOREIGN KEY (a1_R1)
REFERENCES A;
ALTER TABLE B ADD CONSTRAINT FK_a1_R3 FOREIGN KEY (a1_R3)
REFERENCES A;
ALTER TABLE R2 ADD CONSTRAINT FK_a1_R2_1 FOREIGN KEY (a1_R2_1)
REFERENCES A;
ALTER TABLE R2 ADD CONSTRAINT FK_b1_R2_2 FOREIGN KEY (b1_R2_2)
REFERENCES B;
```

Figura 5.31: Ejemplo de script SQL con ALTER TABLE al final para crear las relaciones foráneas.

Para el desarrollo de esta función ha sido de vital importancia contar con tests que han permitido ir iterando y añadiendo funcionalidad asegurándose de que todo seguía funcionando correctamente.

Persistencia

Una funcionalidad bastante importante en este tipo de aplicación es poder contar con una persistencia, que permita recuperar estados de diagramas previamente modelados.

En este caso, se ha incluido a dos niveles:

- Guardar estado de la aplicación web para recuperar el diagrama en proceso modelado.
- Exportar e importar diagramas haciendo uso de la estructura interna.

Persistencia a nivel de navegador

Para lograr esta persistencia se ha creado una función que guarda nuestra estructura interna en el almacenamiento local del navegador.

Almacenamiento	Clave	Valor
▼ Almacenamiento local	diagramData	{"entities":[{"idMx":"2","name":"Entidad","p...
https://draw-entity-relatio	▼ {"entities": [{"idMx: "2", name: "Entidad", position: {x: 31.199996948242188,	
▶ Almacenamiento de sesión	▼ entities: [{"idMx: "2", name: "Entidad", position: {x: 31.199996948242188,	
IndexedDB	▼ 0: {"idMx: "2", name: "Entidad", position: {x: 31.199996948242188, y:	
▶ Cookies	▶ attributes: [{"idMx: "3", name: "Atributo", position: {x: 151.199996,	
Tokens de confianza	idMx: "2"	
Grupos de interés	name: "Entidad"	
▶ Almacenamiento compartido	▶ position: {x: 31.199996948242188, y: 110.4000015258789}	
Almacenamiento en caché	relations: []	
Segmentos de almacenamier		

Figura 5.32: Diagrama interno guardado en el almacenamiento local del navegador

No basta con realizar este proceso, es también necesario crear una función que se ejecute al renderizar el componente por primera vez. Esta función recupera este estado guardado previamente y recrea el diagrama.

Persistencia a nivel de exportar e importar diagramas

Esta funcionalidad se apoya completamente en la anterior. Tan solo se encarga de validar el diagrama previo a su exportación (no se pueden exportar diagramas no válidos) y volcar la estructura interna en un archivo JSON.

Para importarlo, se hace desde un diálogo que pide un fichero. Este fichero debe ser un diagrama en JSON que será validado. Una vez lo tenemos, se guarda el diagrama recuperado en el almacenamiento local del navegador y se fuerza a recrearlo.

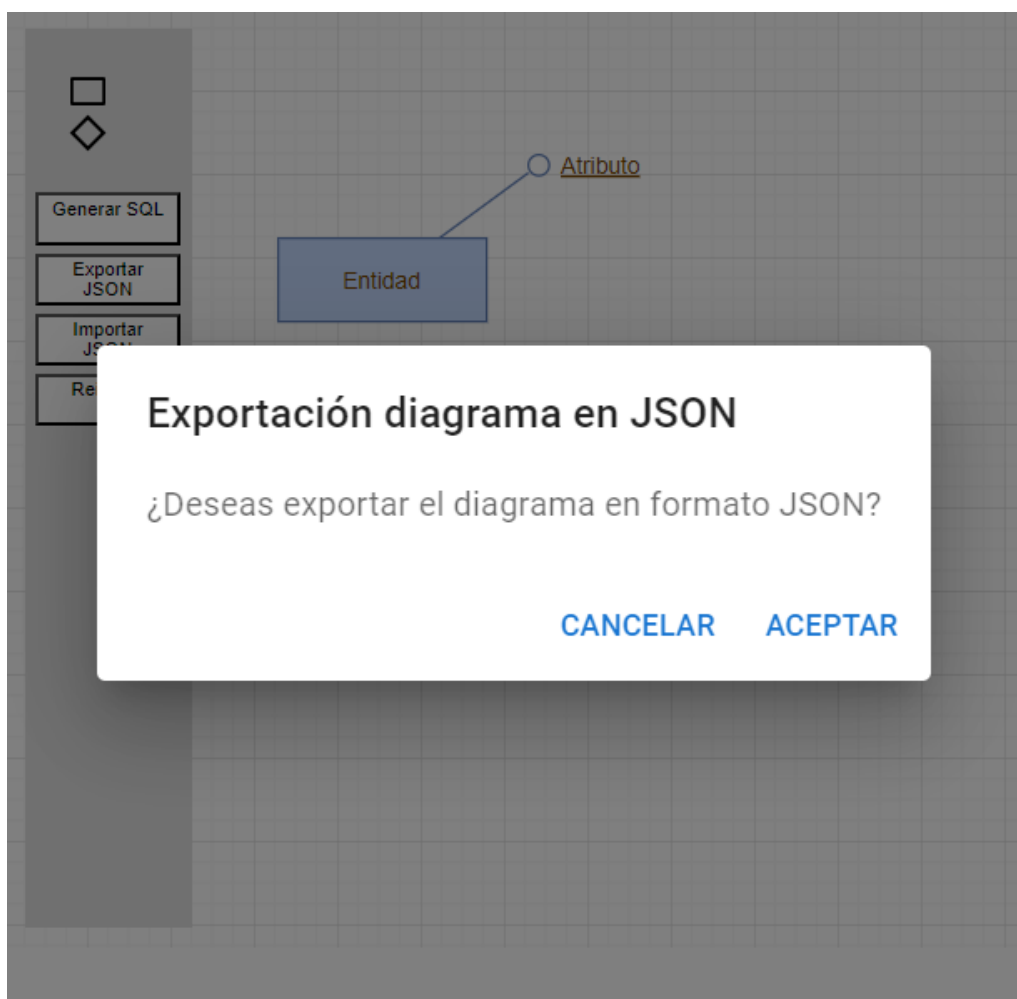


Figura 5.33: Exportar diagrama

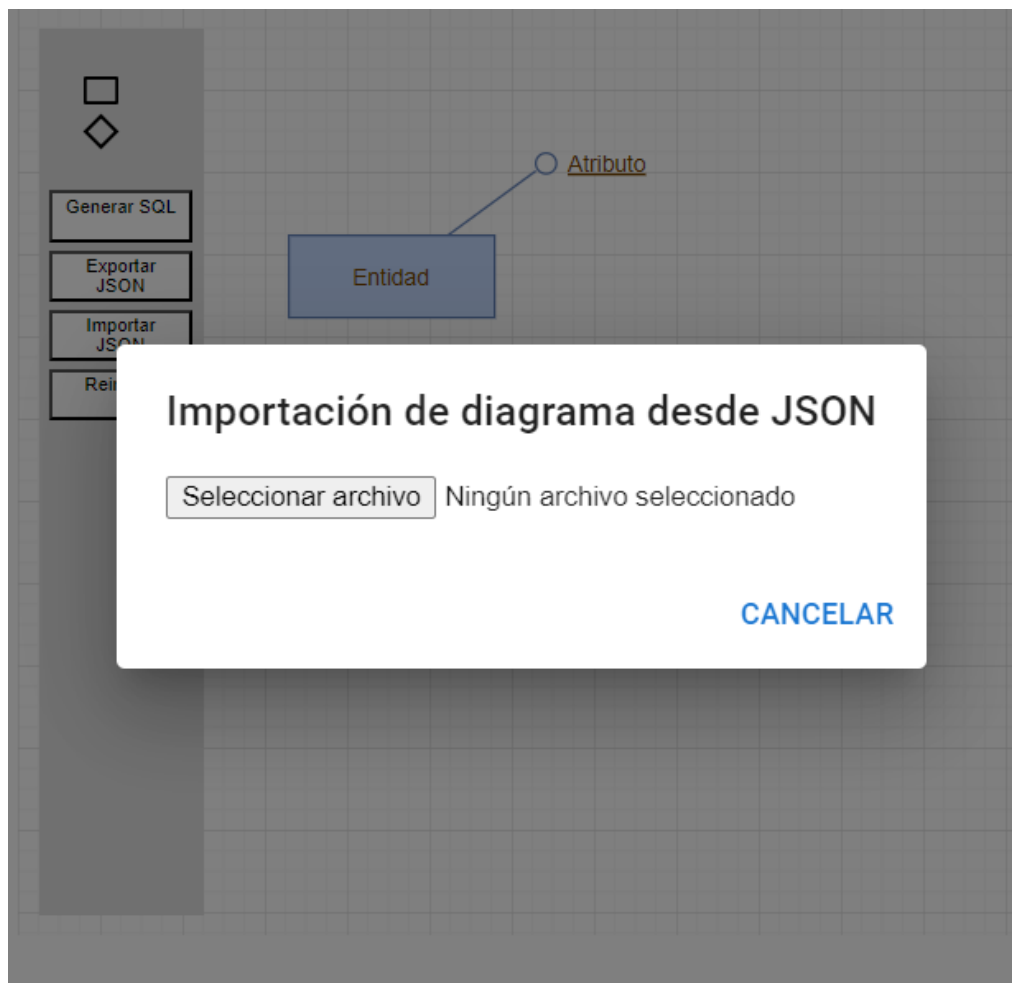


Figura 5.34: Importar diagrama

Retoques finales

Borrado de elementos

Hasta este punto existe un mapeo de teclado para borrar elementos con la tecla Suprimir, el problema es que la borra pero sin reflejar este borrado en nuestro diagrama interno en memoria. Lo cuál supone un problema, por tanto se decidió eliminar este atajo de teclado.

Para esta función nos aseguraremos de que al borrar una entidad no deje atributos, edges, labels o cualquier otro elemento descolgado sin un padre.

Reconfiguración de relaciones

Otro punto a implementar es el de poder reconfigurar los lados de una relación para poder reasignarla a otras entidades sin tener que borrarla y añadirla de nuevo. Requiere retocar la función ya implementada de configuración para eliminar posibles atributos que pudiera tener la relación, edges, cardinalidades ya establecidas o resetear la clave que permite a una relación tener atributos.

Movimiento de objetos

Otro punto que requirió una mejora en su comportamiento es el correspondiente al movimiento de ciertos objetos.

Cuando se mueven entidades, los atributos que están enlazados a estas entidades, no lo hacen con ellas. Esto requirió calcular el offset del atributo con respecto a la entidad, para que al detectar movimiento de una entidad se recalculase la posición de todos sus atributos.

Cuando se crean relaciones reflexivas, y para evitar que los dos lados se solapen en uno solo, se crean puntos intermedios en los edges. El problema es genera al mover la relación, estos puntos quedan estáticos donde fueron creados. La solución fue detectar el movimiento de relaciones reflexivas y recalcular este punto intermedio de los edges para que siempre se mantenga la misma estética.

Resolución de problemas técnicos

A lo largo del proyecto, se enfrentaron varios desafíos técnicos.

Problemas con maxGraph

Como se ha comentado inicialmente se intentó usar maxGraph, que es una librería más moderna y con soporte para los tipos de TypeScript [23] pero debido a las diferencias en su API y la falta de una documentación completa se tuvo que recurrir a mxGraph en su lugar.

Problemas con la versión de Node

La plantilla de mxGraph con React usada tenía una versión de Node antigua (v16.x.x) debido a alguna de sus dependencias. Esto chocaba con el despliegue continuo en Vercel que precisamente estaba descontinuando el soporte a esta versión en junio de este año. Fue necesario revisar las dependencias para poder usar una versión más nueva y estable de Node. Con este cambio también se creó el archivo .node-version que pueden usar algunos de los gestores de versiones de node más populares como fnm [31].

Problemas de transpilación con Jest

Cuando se intentó usar Jest [20] para la creación de tests unitarios en el desarrollo de las funcionalidades de validación del diagrama y generación del SQL se vio que causaba problemas al no poder usar la sintaxis de importación de módulos de Javascript. Al parecer estaba habiendo problemas internos con el transpilador y no era capaz de convertir estos archivos. No fue posible arreglarlo así que se optó por migrar a Vitest [33]. La migración fue directa puesto que comparten API y no fue necesario hacer cambio alguno.

Problemas con mxGraph

Sin duda ha sido mxGraph la parte de la aplicación que más problemas ha dado. Esto es, evidentemente, normal. Se trata del núcleo de la aplicación y la parte con la que más se ha trabajado, es además una librería un tanto antigua lo que se refleja en su API y documentación. Algunos de los problemas concretos han sido:

- Los edges que conectan elementos son, por defecto, reasignables a otros elementos. Lo cuál causaba incoherencias en la aplicación.
- Los edges son, por defecto, dirigidos. Tienen flechas al final.
- Los nombres de los elementos como atributos forman un elemento individual que podía ser movido independientemente del elemento elipse.

Ha sido necesaria bastante investigación en la documentación de la API para saltarse estos comportamientos por defecto y poder adecuar el comportamiento a una aplicación para modelar diagramas entidad-relación.

6. Trabajos relacionados

En este apartado vamos a ver varias herramientas que permiten modelar diagramas. Comentar previamente que no he encontrado ninguna enfocada puramente a generar diagramas entidad-relación.

6.1. Draw.io

Draw.io [6] es una aplicación web que permite modelar diagramas de todo tipo. Es una alternativa realmente potente que no se basa en único tipo de diagramas, pudiendo modelar diagramas de flujo, de secuencias, entre otras muchas opciones. Permite guardar los diagramas en diversos formatos o exportarlos como imágenes.

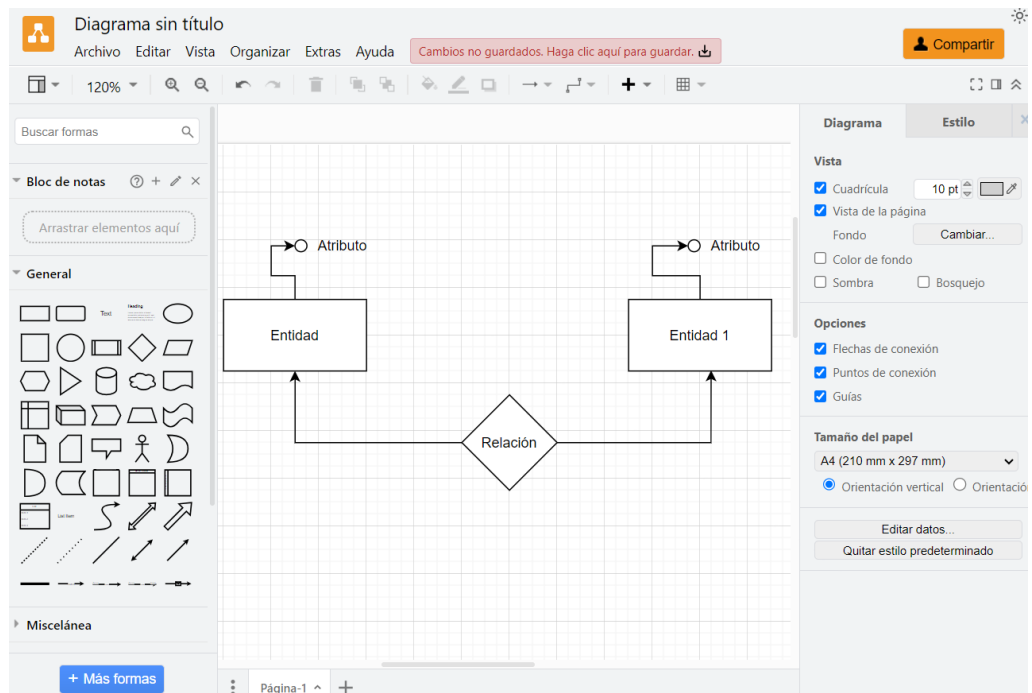


Figura 6.1: Aplicación web Draw.io

6.2. Excalidraw

Excalidraw [8] es una aplicación web que permite modelar diagramas. En este caso su punto diferencial es su estética sketch. Pese a su apariencia es bastante potente y nos permite modelar muchos tipos de diagramas y gráficas. Una de sus característica más destacables es la de "Live Collaboration.^{en} la que crea una sesión compartida donde podemos hacer un diseño colaborativo entre aquellas personas que se conecten a esta sesión.

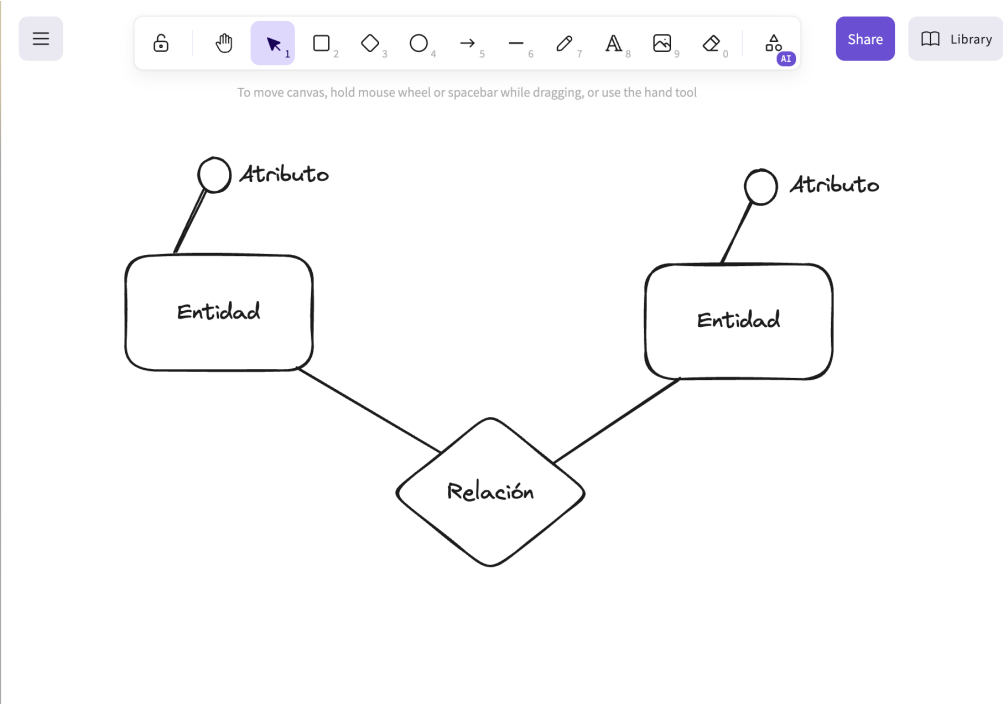


Figura 6.2: Aplicación Excalidraw

6.3. Comparativa frente a alternativas

Se va a hacer una comparativa frente a las alternativas, siempre teniendo en cuenta el caso y objetivos que nos ocupan -modelar un diagrama de entidad relación y generar desde él un script SQL-. La comparativa es evidentemente injusta con alternativas mucho más maduras y potentes para otros cometidos.

Aplicaciones	Draw-E-R	Draw.io	Excalidraw
Modelar diagramas E-R	X	X	X
Validar diagramas E-R	X	-	-
Generar SQL	X	-	-
Exportar e importar	-	X	X

Tabla 6.1: Comparativa del proyecto frente a alternativas

7. Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones

El resultado de la aplicación cumple con todos los objetivos marcados inicialmente. A lo largo del desarrollo, he aprendido técnicas útiles y novedosas para mí como son la división de tareas en sprints o el uso de herramientas de integración continua y de despliegue continuo. La metodología de división de tareas en sprints ha permitido dividir la evolución del desarrollo en tareas más pequeñas y manejables, lo cuál ha facilitado enormemente el desarrollo y su organización. El uso de herramientas de integración y despliegue continuo ha permitido desarrollar estas tareas siempre con la seguridad de mantener un estado de la aplicación correcto. A la vez que el despliegue se realizaba de manera automática, en concreto ha sido extremadamente útil contar con el despliegue preview para recibir feedback del tutor antes de integrar los cambios en la rama principal.

Sin embargo, uno de los mayores desafíos ha sido trabajar con la librería mxGraph. Al tratarse de una biblioteca antigua con una API compleja y que además se encuentra en fin de desarrollo, su uso presentó no pocas dificultades. A destacar que la documentación y su API no es lo intuitiva que sería una librería actual. Los ejemplos que encontramos en internet no suelen ser recientes y eso en el entorno de desarrollo web que varía con carácter mensual es algo muy evidente y problemático. Su sucesora, maxGraph, no estaba lista para ser utilizada debido a que la documentación no estaba completamente migrada e integra varios cambios sobre la API de mxGraph.

A pesar de que la interfaz de usuario cumple con su propósito, considero que es demasiado parca y carece de elementos que la hagan visualmente agradable. Existe un amplio margen de mejora en este aspecto, como por ejemplo el uso de usar Material UI en toda la aplicación y no solo en determinadas partes para establecer un sistema de diseño coherente y agradable.

Además, la aplicación se beneficiaría enormemente de un sistema de usuarios integrado con un backend. Este cambio permitiría a los usuarios guardar sus diagramas en una base de datos (sin necesidad de exportar e importar) y continuar editándolos desde diferentes dispositivos. Objetivo que iría en línea con el marcado inicialmente de crear una aplicación de modelado E-R lo más accesible posible.

Por último, la aplicación podría expandirse para soportar casos más complejos. Como son las entidades débiles, claves y atributos compuestos, relaciones ternarias, agregaciones e ISAs, ofreciendo una herramienta totalmente completa a la hora de modelar diagramas E-R.

En resumen, el balance del proyecto ha sido muy positivo. A pesar de los desafíos técnicos y las áreas identificadas para mejoras, he logrado cumplir con los objetivos iniciales y adquirir habilidades valiosas que serán útiles en presentes y futuros proyectos profesionales.

7.2. Líneas de trabajo futuras

- Pulir determinados comportamientos de la librería: como que los atributos se coloquen unos encima de otros, al mover entidades deberían moverse los atributos en conjunto, cuando se mueven entidades reflexivas deberían calcularse los edges para mantener la estética, etc
- Mejorar la interfaz de usuario para hacerla visualmente atractiva.
- Implementar un sistema de exportación e importación de diagramas para facilitar el trabajo con diagramas existentes.
- Integrar un sistema de usuarios y un backend que permita guardar diagramas en una base de datos, permitiendo a los usuarios continuar editándolos desde diferentes dispositivos.
- Soportar el modelado de casos más complejos como entidades débiles, claves y atributos compuestos, relaciones ternarias, agregaciones e ISAs.

- Poder especificar los tipos de datos de los atributos para que en SQL se generen con los tipos escogidos por el usuario
- Poder dividir el diagrama en vistas, que permitan ver un subsistema y no perderse en un diagrama global que tenga demasiadas entidades.
- Hacer que sea capaz de generar clases persistentes para un ORM (e.g., JPA, SQL-Alchemy)
- Generar diagramas de patas de cuervo o de clases UML a partir del diagrama E-R.

Bibliografía

- [1] Svelte. <https://svelte.dev/>. [Internet; descargado 4-julio-2024].
- [2] Vue. <https://vuejs.org/>. [Internet; descargado 4-julio-2024].
- [3] Atlassian. Gitflow. <https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow>. [Internet; descargado 4-julio-2024].
- [4] Biome. Biome. <https://biomejs.dev/>. [Internet; descargado 4-julio-2024].
- [5] Peter P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.
- [6] Draw.io. Draw.io. <https://app.diagrams.net/>. [Internet; descargado 4-julio-2024].
- [7] EvilMartians. Lefthook. <https://github.com/evilmartians/lefthook>. [Internet; descargado 4-julio-2024].
- [8] Excalidraw. Excalidraw. <https://excalidraw.com/>. [Internet; descargado 4-julio-2024].
- [9] Facebook. <https://es.react.dev/>. <https://es.react.dev/>. [Internet; descargado 4-julio-2024].
- [10] Best Products Finds. What is the difference between a ui/ux designer and a web developer? <https://medium.com/@bestproductfinds/what-is-the-difference-between-a-ui-ux-designer-and-a-web-developer-e14a4fde49de>. [Internet; descargado 4-julio-2024].

- [11] Stan Georgian. What is end-to-end testing and when should you use it? <https://www.freecodecamp.org/news/end-to-end-testing-tutorial/>. [Internet; descargado 4-julio-2024].
- [12] Git. Git documentation. <https://git-scm.com/docs/>. [Internet; descargado 4-julio-2024].
- [13] Git. Git-worktree documentatin. <https://git-scm.com/docs/git-worktree>. [Internet; descargado 4-julio-2024].
- [14] GitHub. Github. <https://github.com/>. [Internet; descargado 4-julio-2024].
- [15] GitHub. Github actions – documentation. <https://docs.github.com/en/actions>. [Internet; descargado 4-julio-2024].
- [16] GitLab. What is continuous integration (ci)? <https://about.gitlab.com/topics/ci-cd/benefits-continuous-integration/>. [Internet; descargado 4-julio-2024].
- [17] IBM. What is continuous deployment? <https://www.ibm.com/topics/continuous-deployment#:~:text=Continuous%20deployment%20is%20a%20strategy,directly%20to%20the%20software's%20users>. [Internet; descargado 4-julio-2024].
- [18] Rubén Maté Iturriaga. Repositorio de draw-er-app en github. <https://github.com/rubenmate/draw-entity-relation>, 2024. [Internet; descargado 4-julio-2024].
- [19] Rubén Maté Iturriaga. Sitio web de draw-er-app. <https://draw-entity-relation.vercel.app/>, 2024. [Internet; descargado 4-julio-2024].
- [20] Jest. Jest. <https://jestjs.io/>. [Internet; descargado 4-julio-2024].
- [21] jGraph. mxgraph. <https://github.com/jgraph/mxgraph>. [Internet; descargado 4-julio-2024].
- [22] maxGraph. maxgraph. <https://github.com/maxGraph/maxGraph>. [Internet; descargado 7-julio-2024].
- [23] Microsoft. Typescript. <https://www.typescriptlang.org/>. [Internet; descargado 4-julio-2024].

- [24] Sayan Mondal. The what, why and how of javascript bundlers. <https://dev.to/sayanide/the-what-why-and-how-of-javascript-bundlers-4po9>. [Internet; descargado 4-julio-2024].
- [25] Mozilla. Dom (document object model) – mdn web docs. <https://developer.mozilla.org/es/docs/Glossary/DOM>. [Internet; descargado 4-julio-2024].
- [26] Mozilla. Learn web development - frameworks and tooling – mdn web docs. https://developer.mozilla.org/en-US/docs/Learn#frameworks_and_tooling. [Internet; descargado 4-julio-2024].
- [27] Mozilla. Spa (single-page application) – mdn web docs. <https://developer.mozilla.org/en-US/docs/Glossary/SPA>. [Internet; descargado 4-julio-2024].
- [28] MUI. Material ui. <https://mui.com/>. [Internet; descargado 4-julio-2024].
- [29] Chidume Nmandi. Transpilers: How they work and how to build your own js transpiler. <https://daily.dev/es/blog/transpilers-how-they-work>. [Internet; descargado 4-julio-2024].
- [30] Playwright. Playwright. <https://playwright.dev/>. [Internet; descargado 4-julio-2024].
- [31] Schniz. fnm - fast node manager. <https://github.com/Schniz/fnm>. [Internet; descargado 4-julio-2024].
- [32] Trunk. What is a linter and why should you use one? <https://trunk.io/learn/what-is-a-linter-and-why-should-you-use-one>. [Internet; descargado 4-julio-2024].
- [33] Vitest. Vitest. <https://vitest.dev/>. [Internet; descargado 4-julio-2024].
- [34] Wikipedia. Javascript — wikipedia, la enciclopedia libre. <https://es.wikipedia.org/wiki/JavaScript>. [Internet; descargado 4-julio-2024].
- [35] Wikipedia. Prueba unitaria — wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Prueba_unitaria. [Internet; descargado 4-julio-2024].

- [36] Wikipedia. Test driven development — wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas. [Internet; descargado 4-julio-2024].