Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

AdAstra: Un Generador Estático de Sistemas de Gestión de Aprendizaje

AdAstra: A Static Learning Management System Generator

Jaime Armas Rivero

La Laguna, 14 de julio de 2023

D. **Casiano Rodríguez León**, con N.I.F. 42.020.072-S profesor Catedrático de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

AdAstra: Un Generador Estático de Sistemas de Gestión de Aprendizaje

ha sido realizada bajo su dirección por D. **Jaime Armas Rivero**, con N.I.F. 79084143T.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 14 de julio de 2023

Agradecimientos

A mi familia y amigos por aconsejarme, fortalecerme y confiar en mí.

Licencia

Resumen

Es un hecho consolidado el uso de GitHub como plataforma para la enseñanza, especialmente en el ámbito de la Informática. La propia empresa apoya estas iniciativas dentro de un grupo de acciones que se engloban bajo el término GitHub Education y que incluye no solo descuentos, foros, congresos, becas, etc. sino también herramientas específicas para la educación como GitHub Classroom, la cual da soporte al proceso de asignación de tareas.

Además, debido a la creciente popularización de las páginas web estáticas basadas en la metodología Jamstack se está extendiendo cada vez más su uso y están saliendo nuevas tecnologías que se respalden en estas.

Este trabajo presenta una herramienta que permita generar LMS estáticos, facilitando el trabajo y configuración de los mismos. Los LMS generados siguen metodologías y prácticas que son comunes en GitHub Education y hacen uso de los servicios y herramientas que proporciona GitHub. La cual se aprovechara de la API GraphQL de GitHub para conectarse con las tareas e información dentro la organización creada por el profesor para impartir sus clases e incorporarlas dentro del LMS.

Palabras clave: github-education, astro, github-classroom, graphql, git, javascript, typescript

Abstract

It is a consolidated fact the use of GitHub as a platform for teaching, especially in the field of Computer Science. The company itself supports these initiatives within a group of actions that fall under the term GitHub Education, which includes not only discounts, forums, conferences, scholarships, etc. but also specific tools for education such as GitHub Classroom, which supports the task assignment process.

In addition, due to the growing popularity of static web pages based on the Jamstack methodology, their use is increasingly widespread and new technologies that rely on them are emerging.

This work presents a tool that allows generating static LMS, facilitating their work and configuration. The generated LMS follows methodologies and practices that are common in GitHub Education and make use of the services and tools provided by GitHub. It will take advantage of the GitHub GraphQL API to connect with the tasks and information within the organization created by the teacher to teach their classes and incorporate them into the LMS.

Keywords: *github-education, astro, github-classroom, graphql, git, javascript, typescript*

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción, Antecedentes y Estado del Arte

1.1. Introducción

En los últimos años, la educación en línea ha cobrado una gran relevancia debido a la pandemia global que obligó a muchos países a implementar medidas de distanciamiento social. Como resultado, muchas instituciones educativas han tenido que adaptarse rápidamente a los sistemas de aprendizaje en línea. Entre estos sistemas, los Learning Management System (LMS) son muy populares debido a su capacidad para ofrecer una plataforma en línea que permite a los profesores compartir recursos y comunicarse con sus alumnos.

En este contexto, `GitHub Education` GitHub, s.f.-a ha surgido como una plataforma líder en la enseñanza de programación y desarrollo de proyectos en equipo. Para ello, se utiliza `GitHub Classroom` GitHub, s.f.-b, una herramienta que permite a los profesores crear tareas y asignaciones de manera automatizada y que se integra perfectamente con GitHub. Sin embargo, una limitación de GitHub Classroom es que solo permite crear tareas y asignaciones que sean entregadas como repositorios de código en GitHub.

Con el objetivo de abordar esta limitación, se propone una aplicación que utiliza la filosofía de GitHub Classroom pero que permite la creación de páginas estáticas para LMS. La aplicación utiliza `GitHub GraphQL` GitHub, 2001 para obtener información de los repositorios y `Astro` Astro, 2023b para generar las páginas estáticas.

1.2. Antecedentes y estado actual del tema

Existen algunas herramientas en línea populares para la creación de páginas web para la enseñanza, como `Moodle` “Moodle - Open-source learning platform | Moodle.org”, s.f. y `Google Classroom` Google, s.f. Sin embargo, estas herramientas están diseñadas para la educación en general y no específicamente para la enseñanza de programación. Además, aunque son gratuitas, pueden requerir una curva de aprendizaje para su uso efectivo.

Por otro lado, `GitLab` GitLab Inc., s.f. y `Bitbucket` Atlassian, s.f. son plataformas populares para la colaboración y gestión de proyectos de programación. Aunque ambos permiten la creación de páginas web para repositorios, su enfoque principal no es la creación de un LMS completo. GitHub Classroom, por otro lado, es una extensión de GitHub que se centra específicamente en la enseñanza de programación. Permite a los profesores crear repositorios y asignaciones para los estudiantes, y ofrece una integración más efectiva con GitHub GraphQL o `GitHub API REST` GitHub, 2022.

La aplicación propuesta utiliza GitHub Classroom, GitHub GraphQL y Astro para

proporcionar una alternativa viable y accesible para la creación de páginas web para LMS de programación en línea.

Astro es una herramienta que permite la creación de páginas web estáticas utilizando componentes de JavaScript. Es fácil de usar y tiene una sintaxis clara y concisa, lo que la hace ideal para la creación de páginas web para LMS. Aunque existen otras herramientas para generación de web estáticas como `Jekyll` Preston-Werner, 2023 o `Hugo` "The world's fastest framework for building websites", 2023, la cuales han sido muy populares en los últimos años, en este caso se ha decidido por usar Astro, un framework nuevo y novedoso.

Esto se debe a que algunas de las principales desventajas de Jekyll y Hugo es que pueden tener una curva de aprendizaje pronunciada para los nuevos usuarios. La sintaxis y la configuración pueden ser complejas para aquellos que no tienen experiencia previa en la creación de sitios web estáticos. Además, a medida que un proyecto se vuelve más complejo, Jekyll y Hugo pueden requerir más tiempo y esfuerzo para mantener y actualizar.

Mientras que las razones por la que se han elegido a Astro han sido su facilidad de uso, su sintaxis clara y concisa, la oportunidad que ofrecer al integrarse con otros frameworks para UI, como `React` Meta, 2022 o AngularGoogle Inc., 2023, su integración directa con la metodología `Jamstack` Netlify, s.f., su fácil personalización a través de plugins, etc.

Además de GitHub GraphQL, GitHub también proporciona una API REST que permite a los desarrolladores interactuar con los repositorios de GitHub. Aunque la API REST no proporciona la misma cantidad de información detallada que GraphQL, sigue siendo una herramienta poderosa para la integración con GitHub.

En comparación con GraphQL, la API REST puede ser más fácil de usar para proyectos más pequeños y simples. También es posible que algunos desarrolladores prefieran la sintaxis y el enfoque de REST sobre GraphQL. Sin embargo, para proyectos más complejos que requieren más información detallada, GraphQL puede ser una mejor opción.

Capítulo 2

Tecnologías

La oferta de tecnologías a la hora de realizar un proyecto software es realmente extensa. A menudo la elección de estas tecnologías para formar la pila de desarrollo está vagamente justificada y el mayor argumento suele ser la familiaridad de los desarrolladores con las herramientas escogidas. Esto, de hecho, no es un argumento menor, ya que en un entorno real, adquirir los conocimientos suficientes sobre una serie de tecnologías como para crear un producto listo para la explotación necesita una gran cantidad de tiempo por parte del equipo de desarrollo y por ende también supone un importante costo económico. Sin embargo, sí que deberían conocerse los puntos débiles y fuertes de las principales opciones y debería realizarse un análisis para evitar problemas graves en el futuro. En este capítulo se describen las tecnologías empleadas para la realización de este proyecto y la justificación de la elección realizada. No se listan dependencias triviales, y que tengan un propósito inmediato (leer ficheros, crear archivos temporales, etcétera).

2.1. GitHub

`GitHub` Microsoft, s.f. es uno de los ejes centrales de todo este trabajo. A pesar de que no sea una plataforma enfocada en la enseñanza, es innegable el valor y productividad que proporcionada a virtualmente todos los desarrolladores de software del mundo. Los profesores pueden gestionar las clases en el mismo ámbito en el que se desarrollan las prácticas o tareas, y con la herramienta desarrollada mejorar la interacción tanto de profesores como de alumnos de los datos guardados en los repositorios de GitHub Classroom.

2.1.1. Las APIs de GitHub: REST y GraphQL

Las `APIs` que nos proporciona GitHub son completamente necesarias. Sin ellas no podríamos tener ningún tipo de comunicación con los servicios de GitHub.

Se proveen dos APIs: `API REST` GitHub, 2022 y `GraphQL` GitHub, 2001. En este trabajo se hace mayormente uso de la API GraphQL. Se ha decidido de esta forma, pues con GraphQL podemos obtener solo los datos que queremos, y podemos recibir en una única llamada información que de otra forma requeriría varías llamadas a diferentes *endpoints*.

2.2. Los Lenguajes JavaScript y TypeScript

El lenguaje de programación escogido es `JavaScript` ECMA International, 2023 (a partir de ahora `JS`), más concretamente su extensión `Typescript` Microsoft, 2023 (a partir de ahora `TS`). Los motivos son variados. Por un lado, es uno de los lenguajes con el que tengo más experiencia y destreza programando. A su vez, si tenemos en cuenta que en el lado de desarrollo web tanto TS como JS es uno de sus mayores pilares es normal entender su uso.

Dejando de lado su popularidad y centrándonos en propiedades más intrínsecas del lenguaje, podemos ver que no hay mejor lenguaje para el manejo de ficheros JSON (usado por las API de GitHub), también al ser un lenguaje interpretado y flexible me ha permitido ser muy productivo en la fase de diseño y prototipado.

Se ha decidido usar TS antes que JS, con el objetivo de minimizar bugs y aprovechar una mejor integración con editores de texto compatibles con: autocompletado inteligente, saltos a referencias, refactorización, etc.

No obstante, TypeScript es un lenguaje compilado y como tal requiere de su propio compilador. Esto añadiría otra dependencia al sistema aparte, aunque a día de hoy muchos frameworks y herramientas proveén de soporte directo de este facilitando el uso a los propios usuarios.

En cuanto a dependencia de desarrollo, se ha usado como gestor de paquetes `pnpm` Pnpm, 2023a ya que se ha planteado usar un monorepo para almacenar todo los paquetes e aplicaciones de la herramienta y a día de hoy pnpm es uno de los mejores gestores de paquetes que se integren con estos.

2.3. El Lenguaje LaTeX y Overleaf

Para la elaboración de este documento se ha usado LaTeX, en específico la implementación de `overleaf` Overleaf, s.f. Gracias a esto hemos podido sincronizar con GitHub, mantener un historial del documento, se ha simplificado la instalación de paquetes y tenemos una copia accesible en línea que se sincroniza automáticamente.

2.4. El sistema de compilación Turborepo

El manejo de un monorepo para almacenar todas las aplicaciones y paquetes relacionados con la herramienta que se está desarrollando puede ser un desafío en términos de gestión de dependencias y versiones. Para abordar este problema, se ha utilizado `TurboRepo` Vercel, 2023, una tecnología que facilita el manejo de monorepos a gran escala.

TurboRepo es una herramienta de gestión de monorepos que se integra con el sistema de control de versiones Git. Proporciona una solución eficiente y escalable para el manejo de dependencias y versiones de múltiples aplicaciones y paquetes que se encuentran en el mismo repositorio. Al usar TurboRepo, se pueden configurar y administrar fácilmente las dependencias entre diferentes aplicaciones y paquetes, y también se puede realizar una gestión más efectiva de las versiones de cada uno de ellos.

Además, TurboRepo proporciona otras funcionalidades, como la capacidad de crear y publicar paquetes, y de gestionar la compilación y el despliegue de las aplicaciones. Todo esto se puede hacer de forma automatizada y con una gran flexibilidad.

En resumen, TurboRepo ha sido una elección clave para el manejo del monorepo que contiene todas las aplicaciones y paquetes relacionados con la herramienta que se está desarrollando. Su capacidad de gestionar dependencias y versiones a gran escala ha sido fundamental para mantener un proceso de desarrollo eficiente y escalable.

2.5. El framework Astro

`Astro` Astro, 2023b es una tecnología de construcción de aplicaciones web que se ha utilizado en este proyecto. Esta herramienta ofrece una solución moderna y rápida para el desarrollo de aplicaciones web basada en la idea de aplicaciones Multi-Páginas (MPA) y cero JS de forma predeterminada, permitiendo una integración fácil y eficiente con otras tecnologías y frameworks.

Una de las principales ventajas de Astro es su capacidad para generar sitios web altamente optimizados y rápidos, esto es gracias a que principalmente es un framework de modelo SSG cuya principal idea es enviar cero JS al cliente a menos que se requiera de dar interactividad, aunque esto no quita que este permita realizar SSR. Además, Astro cuenta con una sintaxis intuitiva y fácil de usar, lo que la hace accesible para los desarrolladores con diferentes niveles de experiencia.

2.5.1. Aplicación de múltiples páginas

Astro usa un arquitectura de `Aplicación de múltiples páginas`"MPAS vs. SPAS", s.f. en el que un sitio web que consta de varias páginas HTML, en su mayoría renderizadas en el servidor. Cuando navegas a una nueva página, tu navegador solicita una nueva página de HTML al servidor.

2.5.2. Soporte de multiples frameworks

Una de las cosas por la que más destaca Astro, es que es una herramienta que esta diseñada para ser completamente personalizable y agnóstica a cualquier framework. Esto permite el uso de componentes de otros frameworks para diseñar la aplicaciones web. Además permite seleccionar si eso componentes se quieren usar solo para añadir diseño a las páginas o para añadir interactividad a través de su arquitectura de `islas de componentes`Astro, s.f.-a.

2.5.3. Arquitectura de islas

Como se menciono anteriormente Astro presenta un modelo de arquitectura de `islas de componentes` Astro, s.f.-a (el cual vamos a llamar `Astro Islands`). Un `Astro Island` se refiere a un componente de UI interactivo en una página HTML predominantemente estática. Pueden existir varias islas en una página, y una isla siempre se renderiza en forma aislada. Piensa en ellos como islas en un mar de HTML estático y no interactivo.

En Astro puedes utilizar cualquier framework de componentes de UI (React, Svelte, Vue, etc.). La técnica en la que se basa este patrón de arquitectura se conoce como hidratación parcial o selectiva a través de directivas que se añaden a los componentes de UI para señalar como quieren ser renderizados. Astro aprovecha esta técnica para hidratar las islas automáticamente.

2.5.4. Integración de la metodología Jamstack

Astro se integra perfectamente con la metodología Jamstack, que se refiere a una arquitectura moderna para la construcción de aplicaciones web que combina una serie de tecnologías, incluyendo JavaScript, APIs, y sitios web estáticos.

En la arquitectura Jamstack, los sitios web se crean como sitios estáticos que se sirven directamente desde una CDN (red de distribución de contenidos). Esto permite una carga rápida y una experiencia de usuario más fluida. Además, el uso de una CDN reduce significativamente la carga en el servidor y disminuye la posibilidad de errores.

Astro se ajusta perfectamente a la metodología Jamstack, ya que proporciona una forma fácil de crear sitios web estáticos utilizando componentes de JavaScript y una sintaxis clara y concisa. Los componentes de Astro se pueden crear de forma independiente y luego se pueden utilizar en diferentes partes del sitio web, lo que hace que la creación de sitios web sea más modular y escalable.

En resumen, Astro es una herramienta ideal para la creación de sitios web estáticos en la metodología Jamstack, ya que proporciona una forma fácil de crear componentes de JavaScript que pueden ser reutilizados en diferentes partes del sitio web. Esto hace que la creación de sitios web sea más modular y escalable, lo que es esencial para una arquitectura Jamstack exitosa.

2.5.5. La sintáxis MDX

De base Astro incluye soporte para archivos `Markdown` Gruber, 2004, los cuales se usan comúnmente para crear contenido con mucho texto, como artículos de blog y documentación. Aunque estos presentán un problema de base y es que si se quiere dejar que el usuario escriba su contenido facilmente y facilitarle le reutilización de material o usar más funcionalidades de las que ya ofrece Markdown. Es por esto que se ha obtado por usar unos de los plugins disponibles en Astro, `MDX` Otander, 2023, el cual combina lo ya existente de Markdown con nuevas funcionalidades como usar sintaxis como JSX o expresiones JS.

2.6. El framework SolidJS

`SolidJS` SolidJS, 2023 ha sido el framework de UI que se ha usado junto a Astro para dar interactividad a la página generada por la aplicación. Este framework esta basado en componentes JSX y su gran ventaja es su enfoque en la reactividad. Utiliza un sistema de reactividad para realizar un seguimiento eficiente de los cambios en el estado de la aplicación y actualizar solo los componentes afectados. Esto significa que cuando los datos cambian, solo se vuelven a renderizar los componentes relevantes, lo que resulta en un rendimiento óptimo y una experiencia de usuario fluida.

SolidJS también ofrece una sintaxis concisa y expresiva para la definición de componentes. Los componentes se definen como funciones y se pueden utilizar hooks para gestionar el estado y los efectos secundarios. Además, SolidJS proporciona una capa de abstracción ligera sobre el DOM, lo que facilita la manipulación y actualización eficiente de los elementos de la interfaz de usuario. Todo esto sin el uso de un DOM virtual,como hacen otros lenguajes como `React` Meta, 2022, sino realizando pequeñas actualizaciones a nodos en el DOM real a través de la reactividad.

Otra característica destacada de SolidJS es su tamaño compacto. El framework tiene un tamaño reducido en comparación con otros frameworks populares, lo que lo convierte en una opción muy atráctiva en este caso, ya que al trabajar con una página estática, la carga de este framework estará directamente en el lado del client, por lo que cuanto más se aligere la carga para este mejor.

2.7. La librería Tailwindcss

`Tailwind CSS` Tailwind Labs, 2023 es un framework de diseño de interfaz de usuario altamente configurable y altamente utilitario. En lugar de proporcionar componentes predefinidos, Tailwind CSS se basa en un enfoque de clases utilitarias, lo que significa que ofrece una amplia gama de clases CSS que se pueden aplicar directamente a los elementos HTML para estilizarlos. Estas clases se utilizan para definir estilos, márgenes, espaciado, colores, tipografía y más.

La principal ventaja de Tailwind CSS radica en su enfoque modular y altamente personalizable. En lugar de depender de un conjunto fijo de componentes, se pueden combinar y reutilizar las clases utilitarias para crear interfaces únicas y adaptables. Esto ofrece una gran flexibilidad y control sobre el diseño de la aplicación.

Además, Tailwind CSS proporciona herramientas adicionales como utilidades de diseño responsivo, clases de animación, control de tamaño de contenedor y más. Estas herramientas facilitan el desarrollo de interfaces adaptables y con un alto rendimiento.

En el TFG, Tailwind CSS se utilizó como el framework principal de estilos para construir la interfaz de usuario de la aplicación. Su enfoque utilitario y su amplia gama de clases facilitaron la personalización y adaptación de los estilos según los requisitos del proyecto.

2.8. El manejador de paquetes Pnpm

`PNPM` Pnpm, 2023a es un administrador de paquetes para proyectos de desarrollo de software en JavaScript. A diferencia de otros administradores, PNPM utiliza un enfoque de almacenamiento global y compartido para las dependencias, lo que evita la duplicación de paquetes y ahorra espacio en disco. Su característica destacada, por la cual se decantó por usarse en este proyecto, es el soporte de `workspaces` Pnpm, 2023b, que permite crear un monorepo, donde múltiples proyectos comparten dependencias comunes, lo que facilita la gestión y el mantenimiento.

PNPM es especialmente útil en entornos de desarrollo que utilizan workspaces y monorepos, ya que reduce la necesidad de descargar y almacenar duplicados de dependencias compartidas. Esto agiliza el proceso de desarrollo al permitir la reutilización de dependencias entre proyectos y garantiza la coherencia de las versiones de las dependencias compartidas.

Capítulo 3

Modo de uso

En este capítulo se explicara una guía detallada paso a paso de como crear y desplegar un LMS con esta aplicación. En los capítulos siguientes se entrará en más detalle con respecto a la toma de decisiones en el diseño y como se ha logrado implementar todo la aplicación.

3.1. Instalar node y un instalador de paquetes de JS

Lo primero será descargar lo necesario para crear una aplicación que use javascript, en este caso, se deberá instalar `node` OpenJS Foundation, 2023, con el siguiente link "Download | Node.js", s.f. para Windows y Mac y en caso de usar una distribución de Linux el siguiente "Installing Node.Js via Package Manager | Node.js", s.f. Esta instalación de normal ya trae incluido el instalor de paquetes npm, el cual se usará para los posteriores ejemplos. Si se desea instalar otro como `pnmn` Pnpm, 2023a o `yarn` Meta et al., 2022, habría que seguir la propia instalación de cada uno.

3.2. Crear un Classroom de GitHub

Esta aplicación esta centrada en la filosofía de enseñanza de GitHub, es por esto que lo primero que se deberá hacer es crear un GitHub Classroom, una herramienta para la enseñanza de GitHub. Para empezar se debera crear una cuenta de GitHub como se especifica aquí "Registrar una nueva cuenta GitHub - documentación de GitHub", s.f. Tras haber creado una cuenta de GitHub se podrá proceder a crear el Classroom, siguiendo la siguiente guía "Administrar aulas - documentación de GitHub", s.f.

3.3. Empezar una LMS con `AdAstra`: create-adastra-lms

Para empezar a usar el generador de LMS se deberá usar el comando `npm create` (Figura 3.1), este comando esta presente en todo los instaladores de paquetes de JS y permite iniciar un proyecto, opcionalmente pasándole el paquete que se quiere usar para la creación. En este caso, se ha creado un paquete para que el usuario pueda iniciar un proyecto con AdAstra con nombre **create-adastra-lms**. Hay que destacar que en Windows debido a que la plantilla del proyecto usa links simbólicos, se deberá usar un terminal con permisos de administrador para ejecutar el comando. Para poder usar este paquete como inicializador se deberá usar el siguiente comando:

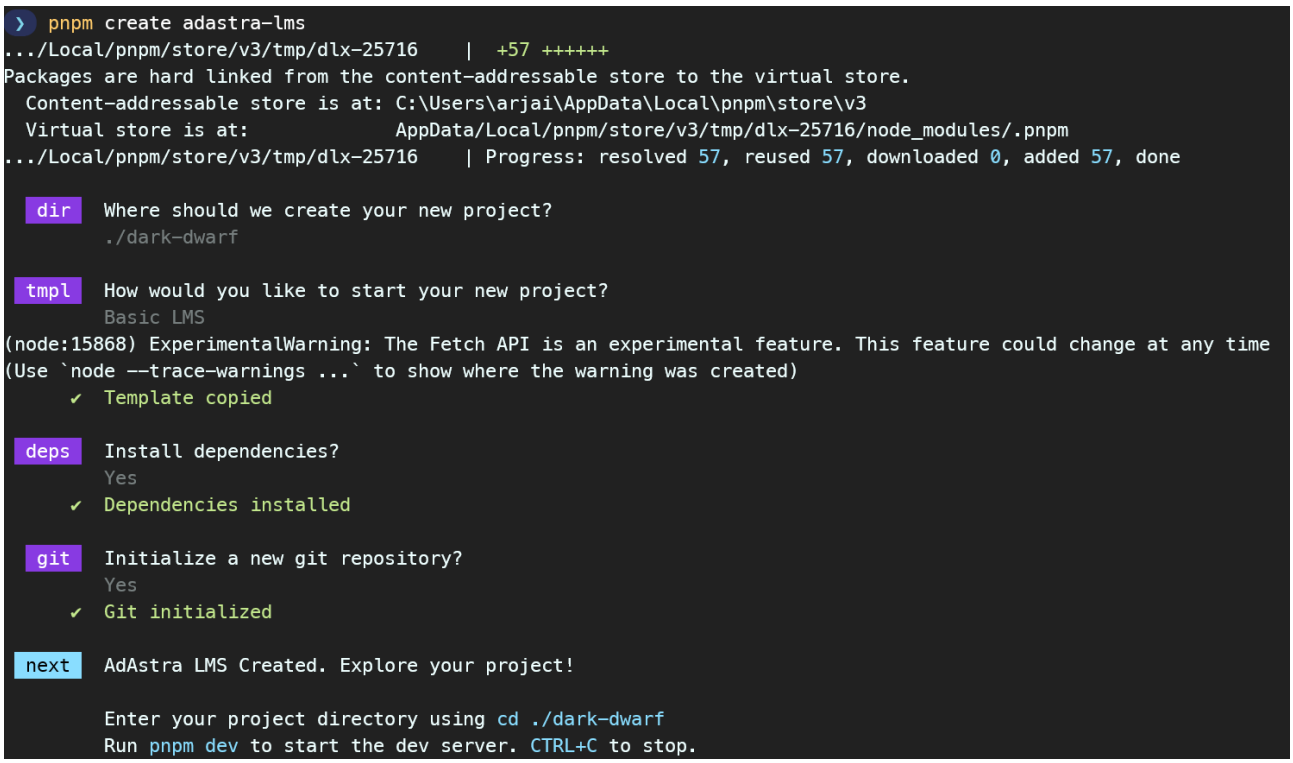

Figura 3.1: Pasos de la ejecución del paquete create-adastra-lms

```
npm create adastra-lms@latest
```

Tras llamar a este comando se iniciará una aplicación de CLI, que preguntará sobre las opciones a querer usar. En este caso, lo que habrá que rellenar será lo siguiente:

1. Directorio de la aplicación.
2. Plantilla a usar. En este caso, solo esta la básica por lo que se deberá usar esa.
3. Si se quiere instalar las dependencias.
4. Si se quiere crear un repositorio de `git`"Git", s.f.

En la figura 3.1 se muestra un ejemplo de este proceso.

Tras usar este comando se creará un proyecto con la estructura mostrada en la figura 3.2.

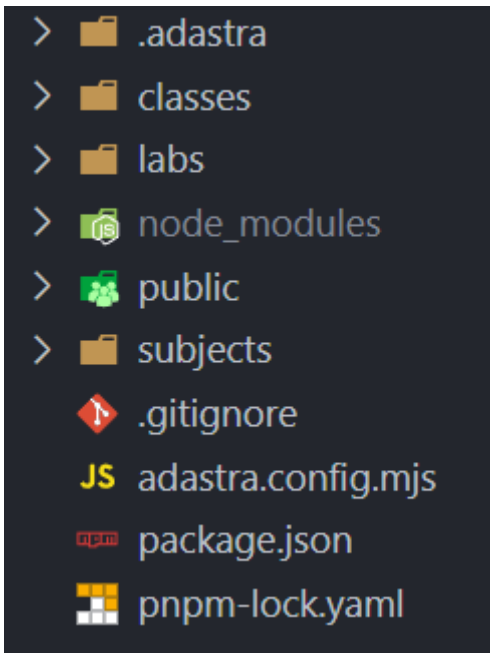

Figura 3.2: Estructura de un proyecto creado con create-adastra-lms

3.4. Configuración del LMS

El primer paso a seguir tras crear el proyecto será configurarlo. Hay dos pasos a seguir, primero actualizar el archivo de configuración de adastra (`adastra.config.mjs`) y segundo añadir las variables de entorno necesarias para el proyecto. Un ejemplo del archivo `adastra.config.mjs` sería el que muestra en la figura 3.3

3.4.1. Archivo de configuración: `adastra.config.mjs`

Para esto habrá que modificar el archivo **adastra.config.js** que se ha creado automáticamente. Dentro de este se podrán encontrar dos objetos, el primero llamado **tailwindConfig** que se puede ver en la figura 3.3 en la linea 1, el cual permitirá cambiar configuraciones relacionadas con el tema de la página que se creará como los colores primarios y secundarios, teniendo cada uno su opción para el estilo oscuro de la página, o incluso el tamaño que asignar al espaciado mínimo o la altura del navbar, etc. Esto funciona con la configuración de tailwindcss como base, en la documentación de tailwindcss ("Theme Configuration - Tailwind CSS", s.f.) se puede encontrar más información sobre esto. Y un segundo, llamado **organizationInfo** que se puede ver en la figura 3.3 en la linea 24, dentro de este habrá que aportar información relacionada con la organización. Entre ellos los datos más importantes a introducir serían los siguientes:

- **pageTitle:** Titulo que aparecerá en la aplicación
- **github:** Objeto de configuración sobre GitHub
  - **organizationName(obligatorio):** El nombre de la organización en GitHub.
  - **classroomUrl:** Url del Classroom creado anteriormente.
- **virtualCampus:** Objeto de configuración sobre el campus virtual

```
export const tailwindConfig = {
  theme: {
    extend: {
      colors: {
        primary: "rgb(51 153 255 / <alpha-value>)",
        "dark-primary": "rgb(51 153 255 / <alpha-value>)",
        secondary: "rgb(135 45 230 / <alpha-value>)",
        "dark-secondary": "rgb(135 45 230 / <alpha-value>)",
        ...
      },
      spacing: {
        "navbar-height": "6rem",
        "sidebar-width": "18rem",
        ...
      },
      fontSize: {
        code: "1rem",
        ...
      },
    },
  },
};

export const organizationInfo = {
  pageTitle: "Test Asignatura",
  github: {
    organizationName: "ULL-ESIT-PL-2122",
    classroomUrl: "",
  },
  virtualCampus: {
    teachingGuideUrl: "",
    campusUrl: "",
    labsUrl: "",
  },
};

```

Figura 3.3: Ejemplo del archivo de configuración de AdAstra: adastra.config.mjs

- **teachingGuideUrl:** Url a la guia docente de la asignatura
- **campusUrl:** Url al campus virtual de la asignatura
- **labsUrl:** Url a las tareas de la asignatura

3.4.2. Configurar las variables de entornos

Además para que la aplicación pueda obtener datos de la organización desde la API GraphQL de GitHub se tendrá que añadir a las variables de entorno del programa un token de acceso personal de GitHub github-token, llamándolo **GITHUB_SECRET**, en la figura3.4 se muestra los `scopes` que necesitará el token. La manera normal de añadir este secreto es a través de un `.env` en el desarrollo y a la hora de hacer el despliegue de la aplicación habría que añadirlo a las variables de entorno del servicio. Por ejemplo, si

se usa `Vercel` Vercel Inc., s.f. como plataforma para el despliegue de la aplicación en la sección de *Settings* hay un apartado llamado *Environment Variables* en el que se pueden añadir variables de entornos a la aplicación. En la figura 3.5 se puede ver de forma más detallada este menú.

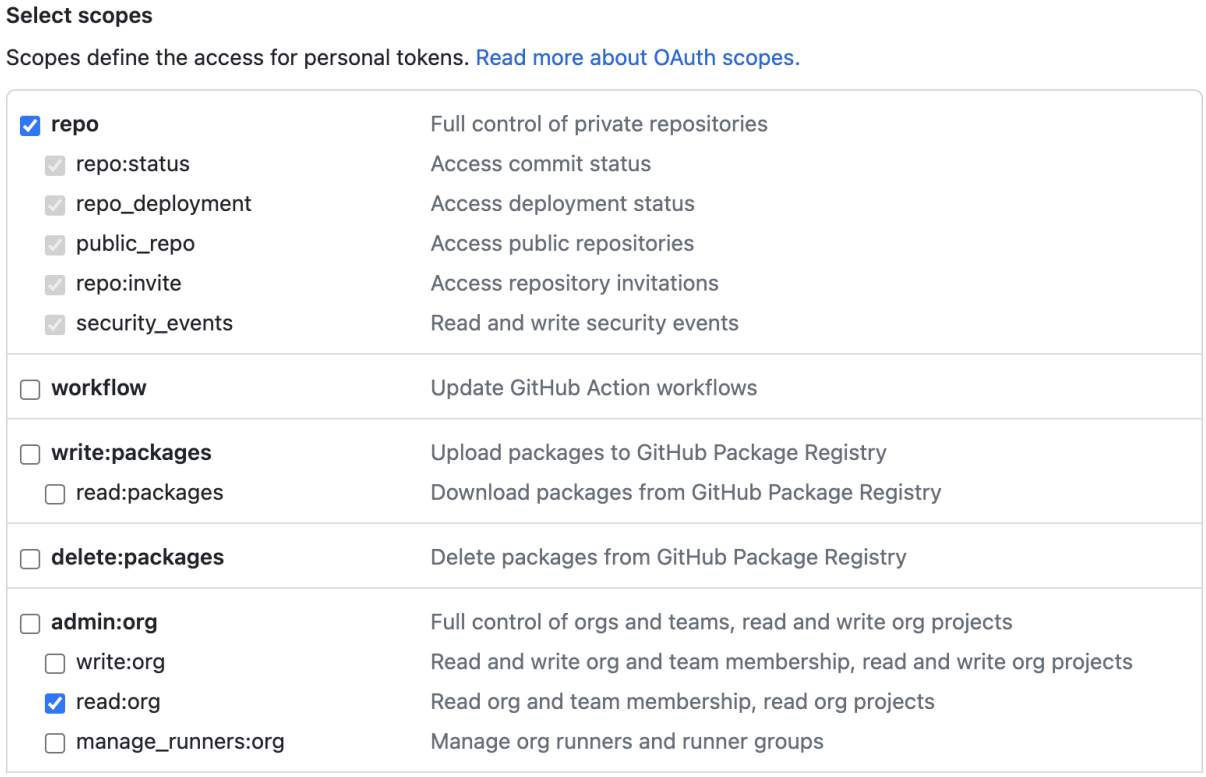

Figura 3.4: Scopes para el token GITHUB_SECRET

Ahora que ya está configurado podremos iniciar el proyecto localmente con el comando:

```
npm run dev
```

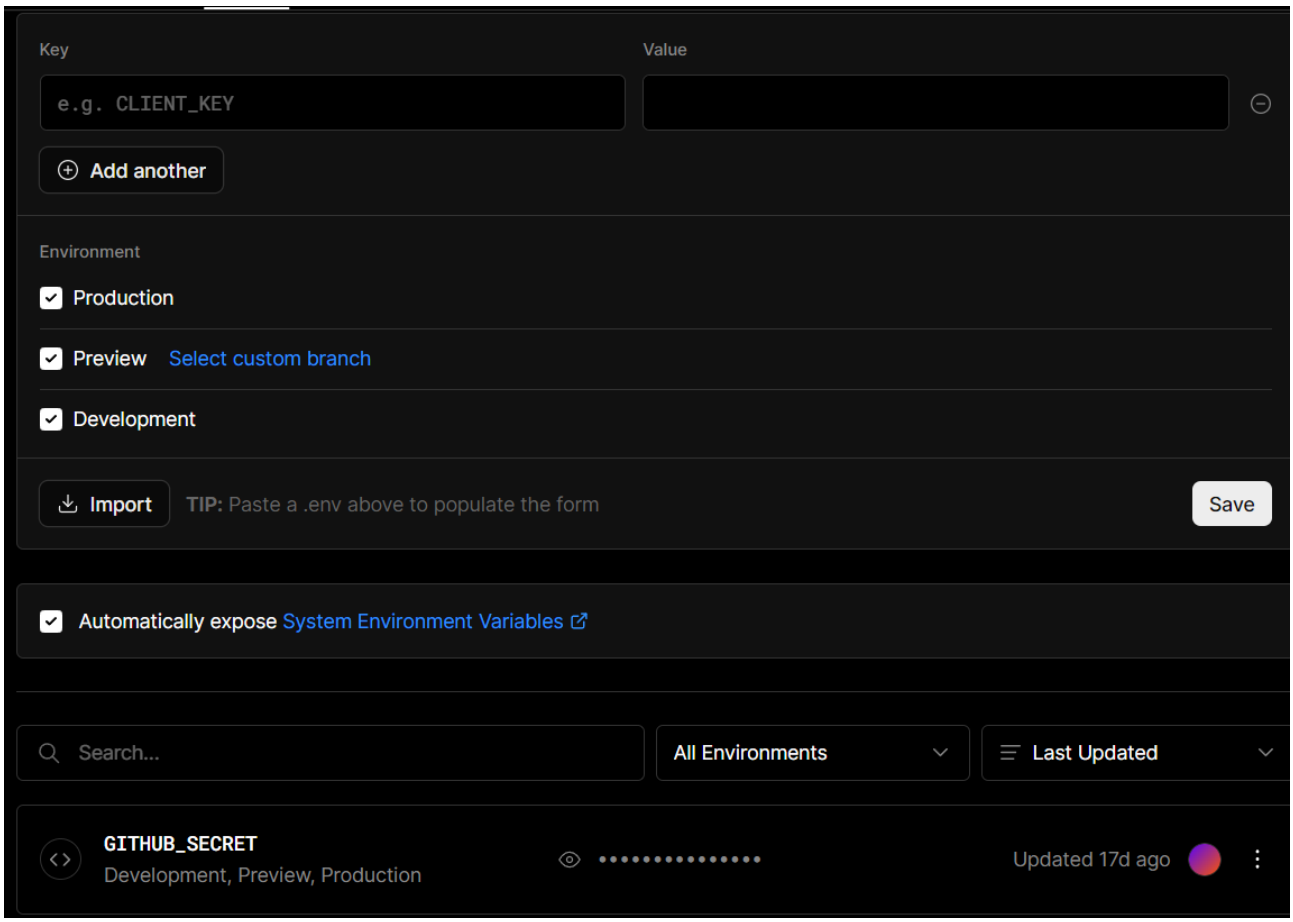

Figura 3.5: Menú para las variables de entorno en Vercel

3.5. Añadir contenido al LMS

En cuanto al contenido habrá que diferenciar en cuatro tipos, elementos generales que se comparten entre todos los contenidos como imágenes, vídeos, archivos, etc; prácticas(`labs`), temarios(`subjects`) y clases(`classes`).

3.5.1. Contenido general: `public`

El primer tipo de contenido, como se dijo anteriormente están relacionados con elementos que se quiera acceder en el resto de manera general, un ejemplo una imagen, y de base no modifican la pagina web que se crea, sino que sirven para expandir el tipo de información a poder usar en los contenidos posteriores. Por esto se proporciona una carpeta llamada **public**. Para poder acceder a estos elementos a posterior, se deberán usar accesos absolutos respecto a la carpeta public. Unos ejemplos serían estos:

```
public/test.png -> /test.png
public/images/test.png -> /images/test.png
```

3.5.2. Añadir un temario

El primer contenido que deberá ser escrito por el usuario directamente y que realizará un cambio explicito en la aplicación serán los temarios, que están ubicados en la carpeta

subjects. Para añadir una nueva entrada se deberá crear un archivo de tipo `Markdown` Gruber, 2004. Se deberá tener en cuenta que la aplicación usa un sistema de navegación automático basado en el nombre de los archivos, por esto se deberá añadir un numero antes del nombre del archivo. Un ejemplo es este:

```
test-subject.md -> 1-test-subject.md
```

Además, el nombre del archivo deberá estar con `-` como separador y tener en cuenta que el nombre del elemento en la navegación irá en función del nombre del archivo.

```
1-test-subject.md -> Test Subject
```

Dentro de estos archivos habrá que añadir contenido como se trabaja de normal con los archivos markdown, aunque se deberá escribir un contenido extra llamado **fromatter** al principio del archivo que permitirá darle información importante al generador. Un ejemplo de como se ve en **subjects** es este:

```
---
type: subject
description: "hello world test"
title: "Getting Started"
key: getting-started
---
```

La explicación de cada uno es la siguiente:

- **type:** En los temarios se deberá poner siempre `subject`.
- **description:** Descripción general del archivo.
- **title:** Será el título que tendrá esa entrada al generarse.
- **key:** Clave con la que se identificará al archivo en el proyecto.

3.5.3. Añadir una práctica

Este contenido funcionará igual que el anterior, pero está relacionado con un tarea o práctica de la asignatura. Hay que destacar que en el frontmatter se proporcionará un *key* para relacionar esta práctica con la creada en el Classroom de Github, esto se usará en la construcción de la información de los alumnos en el apartado de prácticas de cada uno. Un ejemplo de como se ve el frontmatter de los **labs** es este:

```
---
type: lab
title: Egg Parser
key: egg-parser
templateKey: egg-parser-template
---
```

La explicación de cada uno es la siguiente:

- **type:** En las prácticas se deberá poner siempre `lab`.
- **title:** Será el título que tendrá esa entrada al generarse.
- **key:** Clave con la que se identificará al archivo en el proyecto.
- **templateKey:** Clave de la template del lab en la organización de GitHub creado por el Classroom de GitHub que se relacionará con este archivo, en este caso sería el nombre del repositorio de la template.

3.5.4. Añadir una clase

Al igual que los anteriores se usará markdown para añadir una clase. Hay que destacar que en el frontmatter se deberá proporcionar una lista de **labs** y **subjects** relacionados, para que se genere de manera automática los links a estos en la entrada generada en la aplicación. Un ejemplo de como se ve el frontmatter de las **classes** es este:

```
---
type: class
title: Lunes 2023/05/08
relatedLabs:
  - egg-parser
relatedSubjects:
  - getting-started
---
```

La explicación de cada uno es la siguiente:

- **type:** En los clases se deberá poner siempre `class`.
- **title:** Será el título que tendrá esa entrada al generarse.
- **relatedLabs:** Lista de las claves de los labs a relacionar con está clase.
- **relatedSubjects:** Lista de las claves de los subjects a relacionar con está clase.

3.6. Hacer un despliegue del LMS

Para hacer un despliegue se deberá escoger donde querremos que se haga, pero para este ejemplo usaremos el hosting de vercel como ejemplo, aunque el resto de casos deberá ser parecido.

Lo primero será crearse una cuenta de vercel, como tienen para usar GitHub como cuenta usaremos está opción para simplificar el proceso. Tras esto se nos abrirá un panel como el de la figura 3.6, dentro de este seleccionaremos *Add New ->Project*. Esto nos abrirá una nueva página como la figura 3.7, en este caso vercel funciona usando los repositorios de GitHub del usuario como posibles proyectos, deberemos seleccionar el proyecto que queramos desplegar. Esto abrirá un panel para configurar el proyecto como el de la figura 3.8, deberemos introducir los siguientes datos:

- **Project Name:** El nombre que queremos usar en vercel.

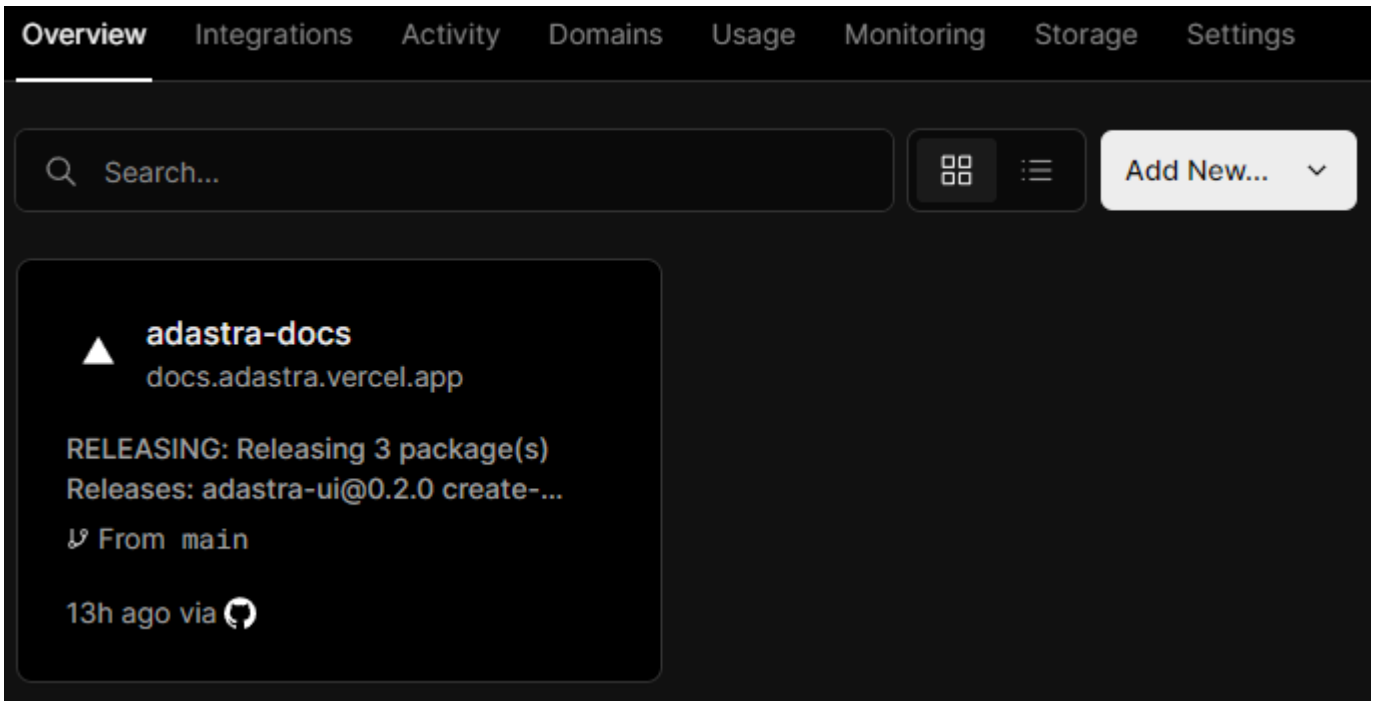

Figura 3.6: Panel principal de Vercel

- **Build Command:** Será el comando que sirva para construir la app:

```
npm run dev
```

- **Enviroment Variables:** Aquí se deberá añadir la variable de entorno `GITHUB_SECRET` con el token de GitHub creado anteriormente.

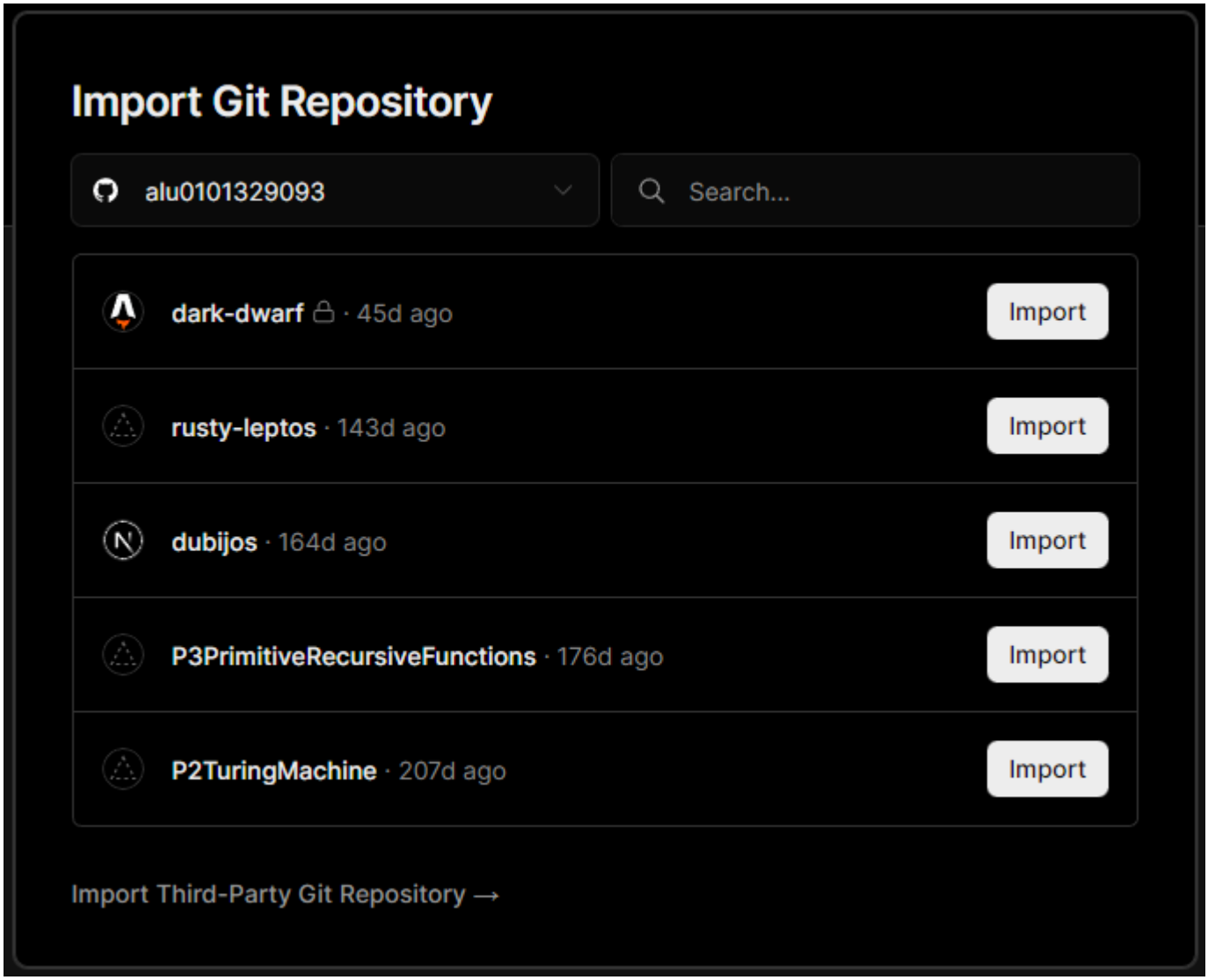

Figura 3.7: Panel para añadir un nuevo proyecto en Vercel

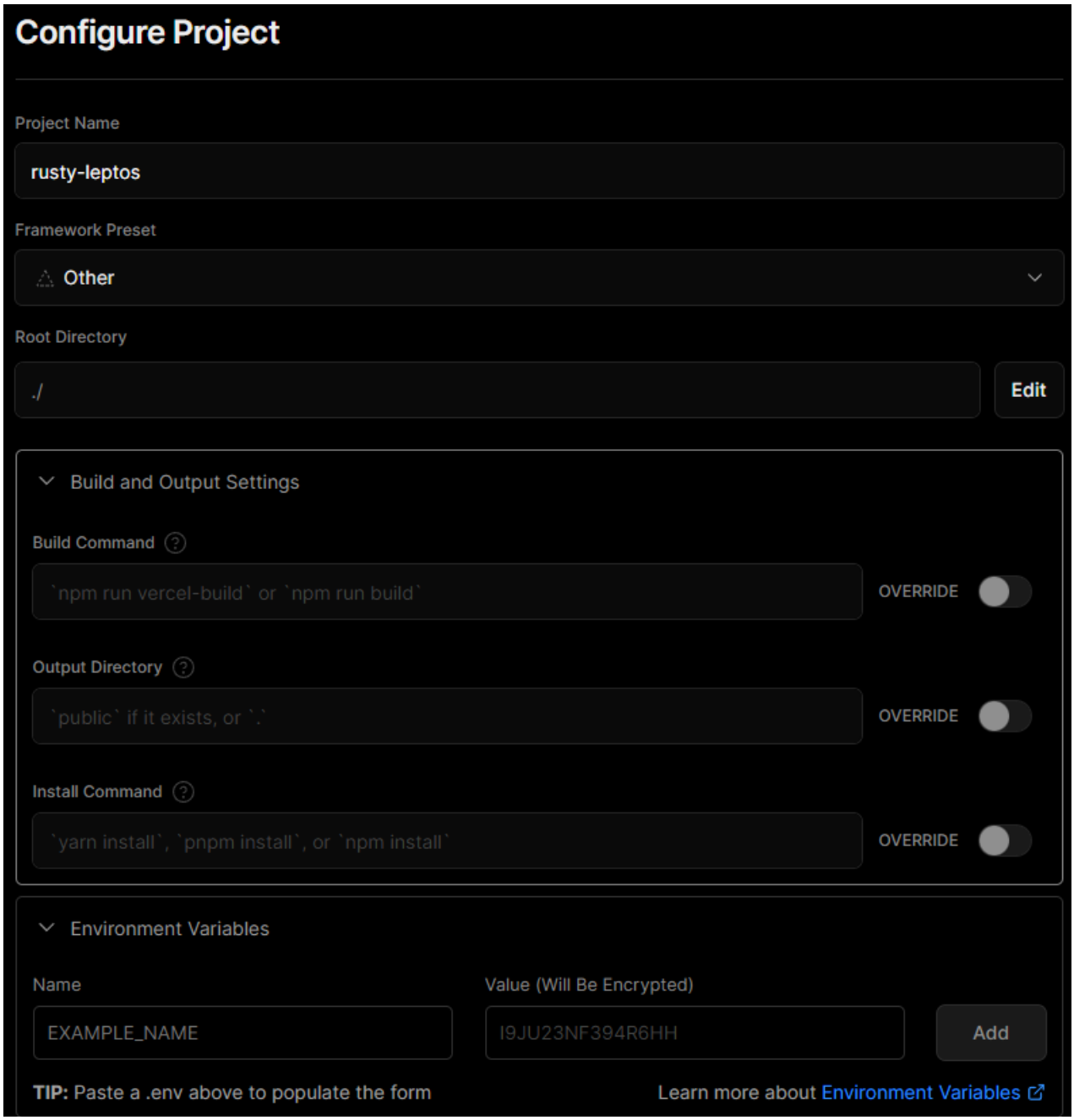

Figura 3.8: Panel de configuración del nuevo proyecto en Vercel

Capítulo 4

Diseño

En este capítulo se explica el porqué se han tomado ciertas decisiones, problemas y dudas que han ido surgiendo a lo largo del desarrollo y se muestra las especificaciones del sistema.

4.1. Consideraciones de Diseño

Cuando se utiliza GitHub en el ámbito académico no hay una sola forma de organizar las clases y queda a criterio de cada profesor. Es por esto que una aplicación inflexible no puede llegar muy lejos. Como resultado, se ha decidido crear una sistema generalista en el que permita al usuario modificar lo máximo posible el resultado final. Todo esto se puede realizar gracias a un archivo de configuración, el cual permite tanto modificar la apariencia de la aplicación como el propio contenido de esta.

El hecho de que ciertos apartados de la aplicación no estén en el ecosistema de GitHub llega a ser un gran problema, ya que no se pueden obtener por la API de GitHub, por esto se permite que los enlaces a esas secciones se puedan introducir a través de la configuración. Por lo que se permitirá cambiar cosas como el nombre de la aplicación y los enlaces a los diferentes contenidos como el google classroom o parecidos, campus virtual de la asignatura, etc.

Otro punto a destacar sería el hecho de que muchas veces un profesor puede querer relacionar ciertos contenidos como las clases, labs o temas entre ellos. Es por esto que para simplificar la experiencia del usuario se les permite introducir en el encabezado de los diferentes archivos markdown, las key de otros archivos para poner enlaces a ellos de manera automática en la página creada.

4.2. El inicializador de `adastra`

Como punto de inicio para usar la aplicación se ha decidido seguir un patrón que siguen la gran mayoría de paquetes en el ámbito de desarrollo de aplicaciones con Javascript y npm. Este se basa en usar el comando `create` de los manejadores de paquetes como npm o yarn. El comando de inicio de adastra sería el siguiente:

```
pnpm create adastra-lms
```

Este comando buscará un paquete con el nombre `create-{name}`, siendo el `name` el parámetro que se le pase al comando, en este caso para adastra sería el paquete

`create-adastra-lms` y se ejecutaría para crear un proyecto la aplicación. Este sistema lo usan la mayoría de framework populares de JS como React o incluso Astro, el framework usado por esta aplicación.

4.3. La template del proyecto

Tras haber ejecutado el comando anterior se creará el proyecto en función de la template que se ha creado para este. Dentro de esta se ha dotado de muchas funcionalidades para simplificar y mejorar la experiencia del usuario.

4.3.1. Creación de contenido por Markdown

El sistema más fundamental sería la creación del contenido principal de la aplicación que en este caso se divide en tres carpetas (labs, classes y subjects), en las cuales se podrá añadir nuevos archivos de tipo markdown con el fin de añadir contenido al LMS.

En este caso el contenido de cada carpeta está pensado para que se use de la siguiente manera:

- **labs**: Comprendería las tareas y laboratorios de la asignatura.
- **classes**: Aquí irían información sobre las clases impartidas. Normalmente separando el contenido en días.
- **subjects**: Por último serían la información relacionadas con los diferentes temaríos de la asignatura. Normalmente habrá una o más clases y labs relacionadas con un subject.

4.3.2. El sistema de navegación en función de la rutas

Uno de los sistemas que más fácilmente se puede ver es que a lo largo del proyecto es que a la izquierda de la web producida se encontraría una barra de navegación que de forma automática. Este sistema se divide en tres partes `tabs`(pestañas), `sections`(secciones) y `entries`(entradas). De base la aplicación está diseñada para que tenga dos `tabs`, en la que cada una tiene múltiples `sections` con uno o más `entries` por cada una. En la figura 4.1 se pueden ver como quedarían las diferente partes del sistema de navegación en el diseño final.

Con esto nos encontramos con un problema y es que de base como se crean las páginas en función de la ruta que tenga, terminaría por ordenar cada parte de la navegación alfabéticamente. Esto le quita flexibilidad a la aplicación, por lo que se decide que el usuario deberá añadir en cada parte de la ruta del archivo un número que simboliza el orden que deba tener usando `-` como separador. Es decir unas rutas como las siguientes:

```
virtual/class/prueba.md
virtual/class/test.md
```

Tendrían un orden alfabético pero al añadir los números para así ordenar quedarían como los siguientes:

```
1-virtual/1-class/1-test.md
1-virtual/1-class/2-prueba.md
```

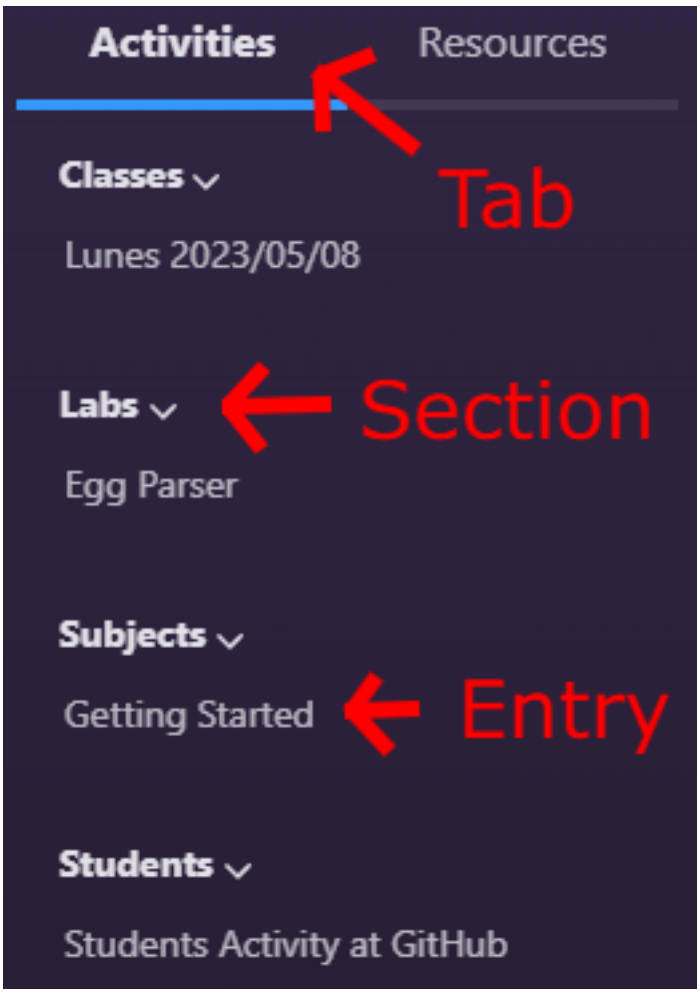

Figura 4.1: Barra lateral con las diferentes partes del sistema de navegación

4.3.3. El fichero de Configuración `adastra.config.mjs`

Uno de los puntos claves al crear está template fue buscar la mayor flexibilidad posible a la hora de poder modificar los contenidos por el usuario, pero sin complicar demasiado el proceso. Es por eso que en un punto se decidió crear un archivo de configuración en que poder especificar ciertos cambios sin tener que modificar el código directamente el usuario.

Dentro de este archivo se estableció dos posibles configuraciones que el usuario quería cambiar:

- **tailwindConfig**: En esta sección se podrá añadir cambios a la propia configuración de la librería de estilos usada por la template para así poder modificar la apariencia de la aplicación de manera sencilla.
- **organizationInfo**: La última sección, pero no menos importante, serviría para añadir nueva información que la aplicación no pueda obtener de la API de GitHub o que se quiera que el usuario pueda decidir que añadir. Algunos ejemplos de esto sería el título que saldría en la aplicación o el link a el campus virtual de la asignatura.

En la figura 3.3 se puede ver un ejemplo de la configuración.

Capítulo 5

Implementación

El objetivo de este capítulo es demostrar las partes importantes o de interés de la implementación y como se ha logrado la realización de lo mencionado en capítulos anteriores.

5.1. El monorepo del proyecto

Una de las cosas fundamentales a la hora de crear este proyecto era conseguir dividir el proyecto en varias secciones sin dificultar el trabajar con ellas a la vez. Es por esto que se decidió usar un monorepo para guardar el proyecto y aprovechar de ciertas herramientas para facilitar su uso. Como herramienta principal para gestionar el monorepo se decidió usar `turborepo` Vercel, 2023, debido a que es uno de las más populares y de los que más rendimiento tienen a la hora de construir las aplicaciones. Además de usar turborepo, se decidió aprovechar una de las opciones que trae el gestor de paquetes `pnpm` Pnpm, 2023a, los `worspaces` Pnpm, 2023b, la cual permite unir varios proyectos dentro del mismo repositorio y compartir las dependencias entre ellos. Por último, se utilizó la herramienta `changesets` Changesets, 2023 para el control de versiones de los diferentes paquetes y proyectos del monorepo.

5.2. El inicializador de `adastra`

Como se dijo anteriormente en el apartado de diseño se ha creado un paquete llamado `create-adastra-lms`, que sirve como punto de inicio para crear un proyecto con AdAstra. La función principal de este paquete se muestra en la figura 5.1. Lo primero que se realiza es la extracción de los argumentos pasados al programa, para luego crear un objeto `context`, que proporciona los datos que necesitará la aplicación, en función de estos argumentos. En caso de que se haya pasado como argumento una *flag* de ayuda, como `-h` o `--help`, se mostrará el menú de ayuda como resultado, se puede ver en la figura 5.2. Para finalizar, se ejecutarán los diferentes pasos de la ejecución, encapsulados dentro de un array, que en este caso se tratan de diferentes funciones asíncronas que se llamarán una después de otra. Para facilitar el proceso de creación de este paquete se ha usado, una librería del framework de `Astro` Astro, 2023b llamada `@astrojs/cli-kit` Astro, 2023a que suministra de diferentes utilidades que sirven para facilitar la creación de un programa de cli, como la petición de un prompt o las personalización de los mensajes.

Ahora vamos a explicar en más profundidad cada parte.

```
1         export const main = async () => {
2           const clearArgv = process.argv.slice(2).filter((arg) => arg !== "--");
3           const context = getContext(clearArgv);
4           if (context.help) return help();
5
6           const steps = [projectName, template, dependecies, git, next];
7
8           for (const step of steps) {
9             await step(context);
10          }
11
12          exit();
13        };
14
```

Figura 5.1: Función principal de create-adastra-lms

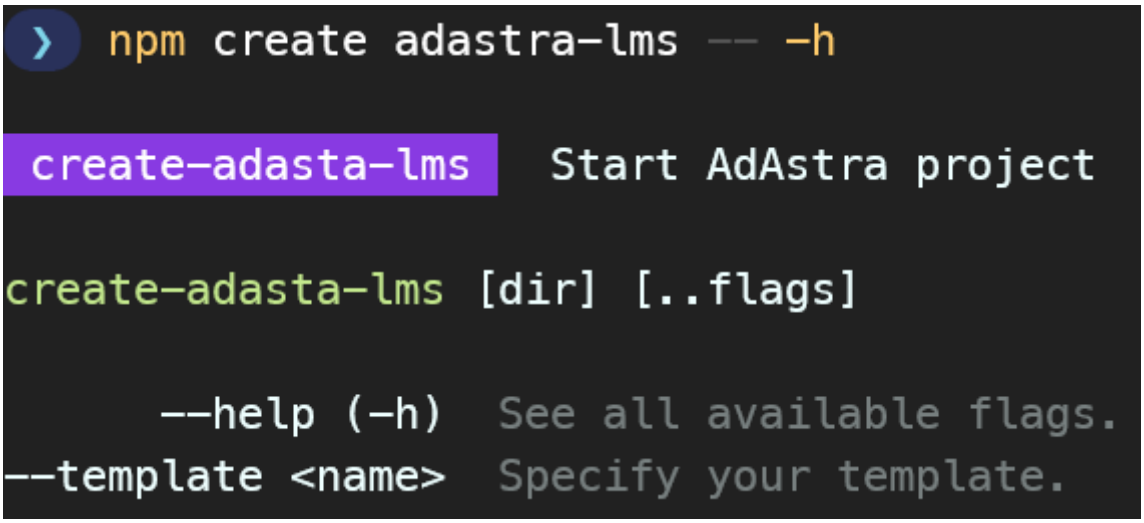

Figura 5.2: Salida del flag de ayuda de create-adastra-lms

5.2.1. La función `getContext`

Está función se encarga de extraer los valores de los argumentos y convertirlo a los datos deseados a través de la librería `arg`Vercel, 2022. Tras se usará el primer parametro pasado como `cwd` y posible nombre del proyecto a crear y se usa la función `detectPackageManager` de la librería `which-pm-runs`zkochan, 2022 para detectar cual manejador de paquetes se está usando y guardar este dato. Con todo esto se construirá un objeto `context` que será el resultado de la función. En la figura 5.3 se puede ver en más profundidad esta función.

5.2.2. El paso `projectName`

Dentro de este paso lo primero que se comprueba es que en caso de si se ha pasado un `cwd` como argumento, este sea valido para usar y este vació. En caso de que la comprobación sea fallida, el programa pedirá al usuario que la ubicación deseada a usar, comprobando que esta sea valida, esta ubicación será el cwd a usar para el proyecto. Tras esto, tanto si el cwd fue valido o no, se extraerá un nombre valido para el proyecto, con la función `toValidName`, que corrigue a un nombre valido el string que se le pase. En la figura 5.4 se ve las partes importantes del código de este paso.

5.2.3. El paso `template`

Este es uno de los pasos más importantes y es que este se encarga de descargar y preparar la *template*, de la cual hablaremos después, a usar para el proyecto. Lo primero que se hace es comprobar si se ha pasado una template como argumento, en caso negativo, se le pide al usuario de entre una lista de templates que escoja la deseada. Tras saber que template se va a querer usar, se empezará el proceso de copia de esta en el que se llamará a la función `copyTemplate` En la figura 5.5 se puede el código de este paso. Dentro de esta primero se intentará descargar la template deseada con la librería `giget`unjs, 2022, que facilita la descarga de proyectos de GitHub a través de un programa de Javascript, a una carpeta llamada `.adastra`, que guardara toda la lógica y funcionamiento del LMS. En caso de que la descarga de un error el programa se parará y mostrará el error. Si todo va bien lo siguiente que se hará es mover el archivo `package.json` que trae la template, que en este caso esta dentro de `.adastra/package.json` a la carpeta principal del proyecto para usarlo como el archivo `package.json` de este proyecto. Tras haber reubicado este archivo se pasará a realizar ciertos cambios a este como cambiar el nombre al del proyecto y cambiar los `scripts` de npm para que seán los siguientes:

```
scripts: {
    dev: 'astro dev --root "./.adastra"',
    build: 'astro build --root "./.adastra"',
}
```

Con estos scripts se define que se deberá usar la carpeta `.adastra` como la carpeta principal a usar por el framework Astro, esto se debe a que para simplificar la experiencia del usuario se decide ocultar la lógica principal del programa de Astro creado en la carpeta `.adastra`, por lo que se deberá usar esa carpeta como `root` del proyecto y no la carpeta principal como normalmente haría el framework. Tras esto se moverá también el archivo `.gitignore` de la template a la carpeta principal, se creará el archivo .env

```
export interface Context {
  help: boolean;
  prompt: typeof prompt;
  cwd: string;
  pkgManager: string;
  projectName?: string;
  template?: string;
  install?: boolean;
  exit(code: number): never;
}

export const getContext = (argv: string[]): Context => {
  const flags = arg(
    {
      "--help": Boolean,
      "--template": String,

      "-h": "--help",
    },
    { argv, permissive: true }
  );

  const pkgManager = detectPackageManager()?.name ?? "npm";
  let cwd = flags["_"][0];
  let { "--help": help = false, "--template": template } = flags;
  let projectName = cwd;

  const context: Context = {
    help,
    prompt,
    pkgManager,
    projectName,
    template,
    cwd,
    exit: (code) => {
      process.exit(code);
    },
  };
  return context;
};

```

Figura 5.3: Función getContext de create-adastra-lms

```
        const checkCwd = async (cwd: string | undefined) => {
          const empty = cwd && isEmpty(cwd);
          if (empty) {
            log("");
            await info(
              "dir",
              `Using ${color.reset(cwd)}${color.dim(" as project directory")}`
            );
          }

          return empty;
        };

        export const projectName = async (context: Context) => {
          await checkCwd(context.cwd);

          if (!context.cwd || !isEmpty(context.cwd)) {
            if (!isEmpty(context.cwd)) {
              await info(
                "Hmm...",
                `${color.reset(`"${context.cwd}"`)}${color.dim(` is not empty!`)}`
              );
            }

            const { name } = await context.prompt({
              name: "name",
              type: "text",
              label: title("dir"),
              message: "Where should we create your new project?",
              initial: `./${generateProjectName()}`,
            });

            context.cwd = name!;
            context.projectName = toValidName(name!);
          } else {
            let name = context.cwd;
            context.projectName = toValidName(extractName(name));
          }

        };

```

Figura 5.4: Paso projectName de create-adastra-lms

para que el usuario pueda introducir la variable de entorno `GITHUB_SECRET` en local y el archivo `adastra.config.mjs` con la configuración necesaria del proyecto. Por último se crearán link simbólicos que permitan modificar el contenido del proyecto de Astro dentro de `.adastra`, pero sin tener que entrar dentro de la lógica de este. Estos serían los siguientes:

- Se vinculará el archivo `.env` creado en la carpeta principal del proyecto con uno dentro de la carpeta `.adastra`, esto es necesario ya que al definir anteriormente que el root a usar a la hora de iniciar el framework de astro fuera la carpeta `.adastra` de normal mirará ahi si se encuentra el archivo `.env` y no fuera de esta.
- Se vinculará las carpetas `classes`, `labs` y `subjects` con las carpetas con sus respectivas contra partes dentro de la carpeta `content` del proyecto de Astro, donde se guarda los contenido markdown a usar. En este caso serían:

```
.adastra/src/content/docs/1-activities/1-classes
.adastra/src/content/docs/1-activities/2-labs
.adastra/src/content/docs/1-activities/3-subjects
```

En la figura 5.6 se puede ver mejor la función `copyTemplate`.

5.2.4. El paso `dependencies`

En el siguiente paso de la ejecución se le preguntará al usuario si se querrá instalar las dependencias ahora, sino se le notificará que deberá hacerlo más tarde. Si se acepta instalar, el programa usará la librería `execa`sindresorhus, 2023, que facilita la ejecución de otros procesos, para llamar el comando `install` junto con el manejador de paquetes, que anteriormente se había detectado y guardado en `context`. En la figura 5.7 se puede ver el codigo de esto.

5.2.5. El paso `git`

Este paso es parecido al anterior, solo que en este caso se pregunta si se quiere iniciar un proyecto de `git`“Git”, s.f., en caso de que no existiera previamente. En caso afirmativo, se usará `execa` al igual que antes para llamar en este caso los siguientes comandos de git que permiten empezar un repositorio:

```
git init
git add -A
git commit -m Initial commit from AdAstra
```

En la figura 5.8 se puede visualizar el código de este paso.

5.2.6. El paso `next`

El último paso es relativamente simple y es que solo se encarga informar al usuario de que el proceso ha terminado y mostrar los posibles paso a seguir:

```
export const template = async (context: Context) => {
  if (!context.template) {
    const { template: tmpl } = await context.prompt({
      name: "template",
      type: "select",
      label: title("tmpl"),
      message: "How would you like to start your new project?",
      initial: "basic",
      choices: [
        {
          value: "basic",
          label: "Basic LMS",
          hint: "(recommended)",
        },
      ],
    });
    context.template = tmpl;
  } else {
    await info(
      "tmpl",
      `Using ${color.reset(context.template)}${color.dim(
        " as project template"
      )}`
    );
  }

  await spinner({
    start: "Template copying...",
    end: "Template copied",
    while: () =>
      copyTemplate(context.template!, context as Context).catch((e) => {
        if (e instanceof Error) {
          error("error", e.message);
          process.exit(1);
        } else {
          error("error", "Unable to clone template.");
          process.exit(1);
        }
      }),
  });
};

```

Figura 5.5: Paso template de create-adastra-lms

```
      const copyTemplate = async (template: string, context: Context) => {
        const templateTarget = `.../${template}`;

        try {
          await downloadTemplate(templateTarget, {
            force: true, provider: "github", cwd: context.cwd, dir: "./.adastra"
          });
        } catch (err: any) {
          throw new Error(err.message);
        }

        fs.renameSync(`${context.cwd}/.adastra/package.json`,`${context.cwd}/package.
  json`);

        const updateFiles = Object.entries(FILES_TO_UPDATE).map(
          async ([file, update]) => {
            const fileLoc = path.resolve(path.join(context.cwd, file));
            if (fs.existsSync(fileLoc)) {
              return update(fileLoc, {
                name: context.projectName!,
                scripts: {
                  dev: 'astro dev --root "./.adastra"',
                  build: 'astro build --root "./.adastra"',
                },
              });
            }
          }
        );

        await Promise.all([...updateFiles]);

        fs.renameSync(`${context.cwd}/.adastra/.gitignore`,`${context.cwd}/.gitignore`)
  ;
        fs.writeFileSync(`${context.cwd}/.env`, "GITHUB_SECRET=\n");

        fs.writeFileSync(
          `${context.cwd}/adastra.config.mjs`,
          `export const tailwindConfig = { ... };

      export const organizationInfo = { ... };
      `);

        fs.symlinkSync(`../.env`, `${context.cwd}/.adastra/.env`, "file");
        symlinkDir(`${context.cwd}/.adastra/public`, `${context.cwd}/public`);
        symlinkDir(`${context.cwd}/.adastra/src/content/docs/1-activities/1-classes`,`$
  {context.cwd}/classes`);
        ...
      };

```

Figura 5.6: Función copyTemplate simplificada del paso template de create-adastra-lms

```
        const install = async ({
          pkgManager,
          cwd,
        }: {
          pkgManager: string;
          cwd: string;
        }) => {
          const installExec = execa(pkgManager, ["install"], { cwd });
          return new Promise<void>((resolve, reject) => {
            setTimeout(() => reject(‘Request timed out after one minute‘), 300_000);
            installExec.on("error", (e) => reject(e));
            installExec.on("close", () => resolve());
          });
        };

        export const dependecies = async (context: Context) => {
          const { deps } = await context.prompt({
            name: "deps",
            type: "confirm",
            label: title("deps"),
            message: ‘Install dependencies?‘,
            hint: "recommended",
            initial: true,
          });
          context.install = deps;

          if (!deps)
            return await info(
              "No problem!",
              "Remember to install dependencies after setup."
            );

          await spinner({
            start: ‘Dependencies installing with ${context.pkgManager}...‘,
            end: "Dependencies installed",
            while: () =>
              install({ pkgManager: context.pkgManager, cwd: context.cwd }).catch(
                (e) => {
                  error("error", e);
                  process.exit(1);
                }
              ),
          });
        };

```

Figura 5.7: Paso dependencies de create-adastra-lms

```
        const init = async ({ cwd }: { cwd: string }) => {
          try {
            await execa("git", ["init"], { cwd, stdio: "ignore" });
            await execa("git", ["add", "-A"], { cwd, stdio: "ignore" });
            await execa(
              "git",
              [
                "commit",
                "-m",
                "Initial commit from AdAstra",
              ],
              { cwd, stdio: "ignore" }
            );
          } catch (e) {
            console.error(e)
          }
        };

        export const git = async (context: Context) => {
          if (fs.existsSync(path.join(context.cwd, ".git")))
            return await info("Nice!", `Git has already been initialized`);

          const { git } = await context.prompt({
            name: "git",
            type: "confirm",
            label: title("git"),
            message: `Initialize a new git repository?`,
            hint: "optional",
            initial: true,
          });

          if (!git)
            return await info(
              "Sounds good!",
              `You can always run ${color.reset("git init")}${color.dim(" manually.")}`
            );

          await spinner({
            start: "Git initializing...",
            end: "Git initialized",
            while: () =>
              init({ cwd: context.cwd }).catch((e) => {
                error("error", e);
                process.exit(1);
              }),
          });
        };

```

Figura 5.8: Paso git de create-adastra-lms

```
AdAstra LMS Created. Explore your project!

Enter your project directory using cd ./name
Run npm run dev to start the dev server. CTRL+C to stop.
```

En la figura 5.9 se puede ver el código de este paso.

5.3. La template del proyecto

Como se explicó en la sección anterior, tras descargar la *template* a usar para el proyecto en la carpeta `.adastra`, la unica manera de interactuar con el proyecto de Astro alojado en esa carpeta es a través de las carpetas con links simbólicos que son `classes`, `labs` y `subjects`, y el archivo de configuración `adastra.config.mjs`. En los siguientes apartados entraremos más en detalle de como se ha implementado el uso de estos, además de algunas de las cosas mencionadas en el diseño de la template.

5.3.1. Creación de contenido por Markdown

Para poder explicar como funciona este sistema lo primero que habrá que destacar son tres funciones importantes de Astro.

- Astro funciona a través de ficheros `.astro`, cuya `sintaxis`Astro, s.f.-b es una combinación de Html puro con Jsx, en la que se podrá insertar código de javascript entre dos `---` al principio del archivo, para a posteriori poder usar lo creado con javascript dentro del html, en la figura 5.10 se puede ver un ejemplo de esta sintaxis.

- Además Astro presenta una característica especial para crear el `routing`Astro, s.f.-d de la aplicación, en la que se considerará un página a construir aquel archivo `.astro` que se encuentre dentro de la carpeta `src/pages` de un proyecto de astro y se usará su ruta relativa respecto a esa carpeta como ruta para la url. Teniendo esto en cuenta, una de las características que tiene este sistema de `file-based routing` son las rutas dinámicas en las que si un archivo presenta un nombre como por ejemplo este `[id].astro`, se crearán múltiples páginas con la misma estructura pero que variaran en función de lo que equivalga a la `id` en la url. Eso si teniendo en cuenta que la aplicación que queremos crear debe ser estática se le deberá pasar a Astro las rutas que queramos que se creen a la hora de construir la aplicación, para esto se deberá exportar una función llamada `getStaticPaths` que devolverá un lista de los parámetros a usar. Además si se añade `...` delante del nombre del parámetro en el nombre del archivo, se capturan también rutas de mayor profundidad. En la figura 5.11 se puede ver un ejemplo de las rutas dinámicas.

- Por último sería su sistema de `colecciones de contenido`Astro, s.f.-c, el cual permite añadir colecciones de archivos Markdown, creando carpetas dentro de la ubicación `src/content`, con el añadido de poder configurar y agregar tipos a las colecciones, y es que Astro extiende el funcionamiento de Markdown permitiendo añadir un frontmatter, zona delimitada por dos lineas con los símbolos `---` en la parte superior del archivo con un formato parecido al de los archivo `Yaml`Red Hat, 2023, dentro de este se podrá agregar parámetros que Astro podrá leer y usarse en el código. Un ejemplo de un fromatter sería el de la figura 5.12.

```
export const nextSteps = async ({
  projectDir,
  devCmd,
}: {
  projectDir: string;
  devCmd: string;
}) => {
  const max = stdout.columns;
  const prefix = max < 80 ? " " : " ".repeat(9);
  await sleep(200);
  log(
    `\n ${color.bgCyan(` ${color.black("next")} `)}  ${color.bold(
      "AdAstra LMS Created. Explore your project!"
    )}`
  );

  await sleep(100);
  if (projectDir !== "") {
    projectDir = projectDir.includes(" ")
      ? `"./${projectDir}"`
      : `./${projectDir}`;
    const enter = [
      `\n${prefix}Enter your project directory using`,
      color.cyan(`cd ${projectDir}`, ""),
    ];
    const len = enter[0].length + stripAnsi(enter[1]).length;
    log(enter.join(len > max ? "\n" + prefix : " "));
  }
  log(
    `${prefix}Run ${color.cyan(devCmd)} to start the dev server. ${color.cyan(
      "CTRL+C"
    )} to stop.`
  );
  await sleep(100);
};

export const next = async (context: Context) => {
  let projectDir = path.relative(process.cwd(), context.cwd);
  const devCmd =
    context.pkgManager === "npm" ? "npm run dev" : `${context.pkgManager} dev`;

  await nextSteps({ projectDir, devCmd });
};

```

Figura 5.9: Paso next de create-adastra-lms

```
---
const name = "Astro";
---
<div>
  <h1>Hello {name}!</h1>  <!-- Outputs <h1>Hello Astro!</h1> -->
</div>

```

Figura 5.10: Ejemplo de la sintaxis de Astro

```
---
export function getStaticPaths() {
  return [
    {params: {dog: 'clifford'}},
    {params: {dog: 'rover'}},
    {params: {dog: 'special/spot'}},
  ];
}

const { dog } = Astro.params;
---
<div>Good dog, {dog.replace("/", " ")}!</div>

```

Figura 5.11: Ejemplo de las rutas dinámicas de Astro

```
---
title: "Test 1"
relatedIds:
- 1
- 2
---

```

Figura 5.12: Ejemplo del frontmatter de los archivos Markdown en Astro

```
export const getStaticPaths = async () => {
  const allPages = await getCollection("docs");
  return allPages
    .map((page) => {
      return { params: { page.slug }, props: page };
    })
    .filter((value) => value != undefined);
};

```

Figura 5.13: Ejemplo de la función getStaticPaths usada para la creación del contenido

Con estás funcionalidades básicas de Astro explicadas podemos pasar a ver como se han usado para crear el sistema de creación de contenido por Markdown. Lo primero que se hizo fue crear una nueva colección llamada `docs` donde se guardará en el futuro todos los markdown de la app. Con está colección creada pasamos a crear un archivo en `src/pages` llamado `[...slug].astro`, que recibirá cualquier ruta desde la ruta principal de la web. Dentro de este archivo se creará la función `getStaticPaths`, en la que se hará uso de la función de Astro `getCollection`, que permite conseguir una lista con datos importantes de cada archivo markdown en la colección deseada, entre los datos de cada entrada en este caso el que nos interesa es `slug`, que justo coincide con el nombre del parámetro del archivo creado, este simbolizada la ruta que tendría el archivo como si fuera para una url. Esto es justo lo que queremos para identificar cada entrada estática a querer crear con el archivo `[...slug].astro`, por lo que pasaremos el `slug` obtenido como parámetro de cada entrada y además guardaremos para cada entrada los datos que se consiguieron para cada una con `getCollection` y los guardaremos como `props` de la página para así poder hacer uso de estos datos más tarde. Con todo esto lo que hemos conseguido es que para cada archivo markdown creado dentro de la colección `docs`, se cree automáticamente un página nueva en la aplicación con esa misma ruta. En la figura 5.13 se puede ver un ejemplo parecido a la función `getStaticPaths` usada.

Ahora lo único que faltaría es añadir el Html que quiera que se muestre por cada entrada de la aplicación. Para esto lo primero que haremos será crear un `Layout` que se encargará de añadir estilo a toda las páginas y mostrar funcionalidades como el sistema de navegación, del que hablaremos posteriormente. Con este layout hecho, solo faltaría mostrar el contenido del propio archivo markdown al que está asociado cada página, para ello aprovecharemos que anteriormente se paso los datos de cada markdown como `props` de las páginas para obtener la función `render`, que convierte el contenido markdown a un componente de Astro que usaremos en la página. Con esto ya se puede acceder al contenido de cada archivo markdown de la app. En la figura 5.14 se puede ver un ejemplo que muestra lo explicado.

5.3.2. El sistema de navegación en función de la rutas

Como se dijo anteriormente en el diseño el LMS incorpora un sistema de navegación incorporado para poder movernos entre los diferentes contenidos de la aplicación. En este caso como se dijo en el apartado anterior el contenido se encuentra en una colección de archivos markdown llamada `docs`, por lo que el sistema de navegación también hará

```
---
...
export type Props = CollectionEntry<"docs">;
const { data, render } = Astro.props;
const { Content } = await render();
---

<BaseLayout data={data}>
  <Content />
</BaseLayout>

```

Figura 5.14: Ejemplo con el html del archivo [...slug].astro

```
export type Entry = {
  text: string;
  slug: string;
};

export type Section = {
  name: string;
  entries: Entry[];
};

export type NavegationTab = {
  type: string;
  sections: Section[];
};

```

Figura 5.15: Tipos que representan la navegación

uso de del sistema de colecciones de Astro. Ahora en cuanto a lo visto anteriormente del apartado de diseño en nuestra implementación `entry`, `section` y `tab` sería la mostrada en la figura 5.15

Teniendo en cuenta estos tipos lo que se creo fue una función encargada de obtener la colección `docs` con `getCollection`. Ahora de estos datos extraemos solo lo que vayamos a necesitar que sería el `slug`, separando cada parte de la ruta en función de `/`, y `title` de cada entrada. Esto quedaría como lo msotrado en la figura 5.16

Teniendo ahora para cada archivo markdown su ruta separada en sus respectivas partes, podemos empezar a construir un objeto de tipo `NavigationTab` usando las diferentes partes de cada ruta. Para esto crearemos un objeto llamado `navigationWithId`, en el que guardaremos los diferentes `NavegationTab` que se vayan consiguiendo. Además como se comento en el apartado de diseño, se le pide al usuario que introduzca un número delante de cada parte de la ruta para así añadir un orden a cada parte, por esto se añade un parámetro `id` temporal a `Entry`, `Section` y `NavegationTab` en el que se extraerá el

```
export const getNavigation = async (): Promise<NavegationTab[]> => {
  if (navegationTabs != undefined) return navegationTabs;

  const allPages = await getCollection("docs");
  const routes = allPages.map(({ slug, data }) => ({
    route: slug.split("/"),
    title: data.title,
  }));
...
}

```

Figura 5.16: Primera parte de la función getNavigation

número añadido en la ruta para definir el orden. Para hacer esto, se creará otra función que dividirá un texto en `id` y `text`, y otra función para cada parte que se encargue de ver si esa parte de la ruta con esa id ya está agregada, si no la crea. Estas funciones se pueden ver en la figura 5.17 y como seguiría la función `getNavigation` en la figura 5.18.

Con el objeto `navigationWithId` ya construido lo único que queda usar las ids para ordenar los arrays y eliminar las ids temporales que ya no se necesitan. En la figura 5.19 se puede ver como se ordena los arrays y se finaliza la función `getNavigation`.

5.3.3. Información de los estudiantes

Una de las páginas que se le dan hechas al usuario es una llamada `students`, está se encargará de recoger los datos de la organización de GitHub, para construir una lista de cartas que muestran información relevante de cada estudiante y sus diferentes labs. Para esto se hacen diferentes peticiones a la API GraphQL de GitHub, encapsuladas en varias funciones. Por ejemplo hay una función que se encarga de obtener la información básica de cada estudiante como su nombre, el link a su GitHub su foto de usuario, etc, se puede ver en la figura 5.20. Quedando como resultado en la página algo parecido a lo que se puede ver en la figura 5.21

```
    const getTabIndex = (
      navigationWithId: NavegationTabWithId[], tabId: number, tabText: string
    ): number | undefined => {
      if (Number.isNaN(tabId)) return;
      const tabIndex = navigationWithId.findIndex(
        (value) => value.type === tabText
      );
      if (tabIndex !== -1) return tabIndex;
      navigationWithId.push({ id: tabId, sections: [], type: tabText });
      return navigationWithId.length - 1;
    };

    const getSectionIndex = (
      navigationWithId: NavegationTabWithId[], tabIndex: number,
      sectionId: number, sectionText: string
    ): number | undefined => {
      if (Number.isNaN(sectionId)) return;
      const sectionIndex = navigationWithId[tabIndex].sections.findIndex(
        (value) => value.name === sectionText
      );
      if (sectionIndex !== -1) return sectionIndex;
      navigationWithId[tabIndex].sections.push({
        id: sectionId, entries: [], name: sectionText,
      });
      return navigationWithId[tabIndex].sections.length - 1;
    };

    const getEntryIndex = (
      navigationWithId: NavegationTabWithId[], tabIndex: number, sectionIndex: number,
      entryId: number, entrySlug: string, entryText: string
    ): number | undefined => {
      if (Number.isNaN(entryId)) return;
      const entryIndex = navigationWithId[tabIndex].sections[
        sectionIndex
      ].entries.findIndex((value) => value.slug === entrySlug);
      if (entryIndex === -1)
        navigationWithId[tabIndex].sections[sectionIndex].entries.push({
          id: entryId, slug: entrySlug, text: entryText,
        });
      return sectionIndex;
    };
```

Figura 5.17: Funciones encargadas de buscar o crear las partes de una ruta

```
export const getNavigation = async (): Promise<NavegationTab[]> => {
...
  const navigationWithId: NavegationTabWithId[] = [];
  routes.forEach(({ route, title }) => {
    if (route == undefined) return;
    if (route.length < 2 || route.length > 3) return;

    const [tab, section, entry] = route;
    if (entry != undefined) {
      const { id: tabId, text: tabText } = extractIdAndText(tab);
      const tabTitleText = kebabCaseToTitleCase(tabText);
      const tabIndex = getTabIndex(navigationWithId, tabId, tabTitleText);
      if (tabIndex == undefined) return;

      const { id: sectionId, text: sectionText } = extractIdAndText(section);
      const sectionTitleText = kebabCaseToTitleCase(sectionText);
      const sectionIndex = getSectionIndex(
        navigationWithId, tabIndex, sectionId, sectionTitleText
      );
      if (sectionIndex == undefined) return;

      const { id: entryId, text: entrySlug } = extractIdAndText(entry);
      const entryText = title;
      getEntryIndex(
        navigationWithId, tabIndex, sectionIndex, entryId,
        `/${tabText}/${sectionText}/${entrySlug}`, entryText
      );
    }
  });
...
}

```

Figura 5.18: Segunda parte de la función getNavigation

```
1    export const getNavigation = async (): Promise<NavegationTab[]> => {
2    ...
3      navegationTabs = navigationWithId
4        .sort((firstNav, secondNav) => firstNav.id - secondNav.id)
5        .map(({ sections, type }) => ({
6          type,
7          sections: sections
8            .sort(
9              (firstSection, secondSection) => firstSection.id - secondSection.id
10           )
11           .map(({ entries, name }) => ({
12             name,
13             entries: entries
14               .sort((firstEntry, secondEntry) => firstEntry.id - secondEntry.id)
15               .map(({ id, ...entry }) => entry),
16           })),
17       }));
18
19     return navegationTabs;
20   };
21
```

Figura 5.19: Parte final de la función getNavigation

```
export const getUsers = async (roleFilter?: string[]): Promise<UserInfo[]> => {
  const response = await fetch("https://api.github.com/graphql", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Authorization: `BEARER ${import.meta.env.GITHUB_SECRET}`,
    },
    body: JSON.stringify({
      query: `
        query ($org: String!, $after: String) {
          organization(login: $org) {
            membersWithRole(first: 100, after: $after) {
              edges {
                node {
                  login
                  name
                  avatarUrl
                  url
                }
                role
              }
            }
          }
        }
      `,
      variables: {
        org: organizationName,
      },
    }),
  });
  const { data }: UsersQuery = await response.json();
  if (roleFilter == undefined)
    return data.organization.membersWithRole.edges.map(({ node }) => node);
  return data.organization.membersWithRole.edges.filter((edge) => roleFilter.includes
(edge.role)).map(({ node }) => node);
};

```

Figura 5.20: Query GraphQl con la información básica de los estudiantes

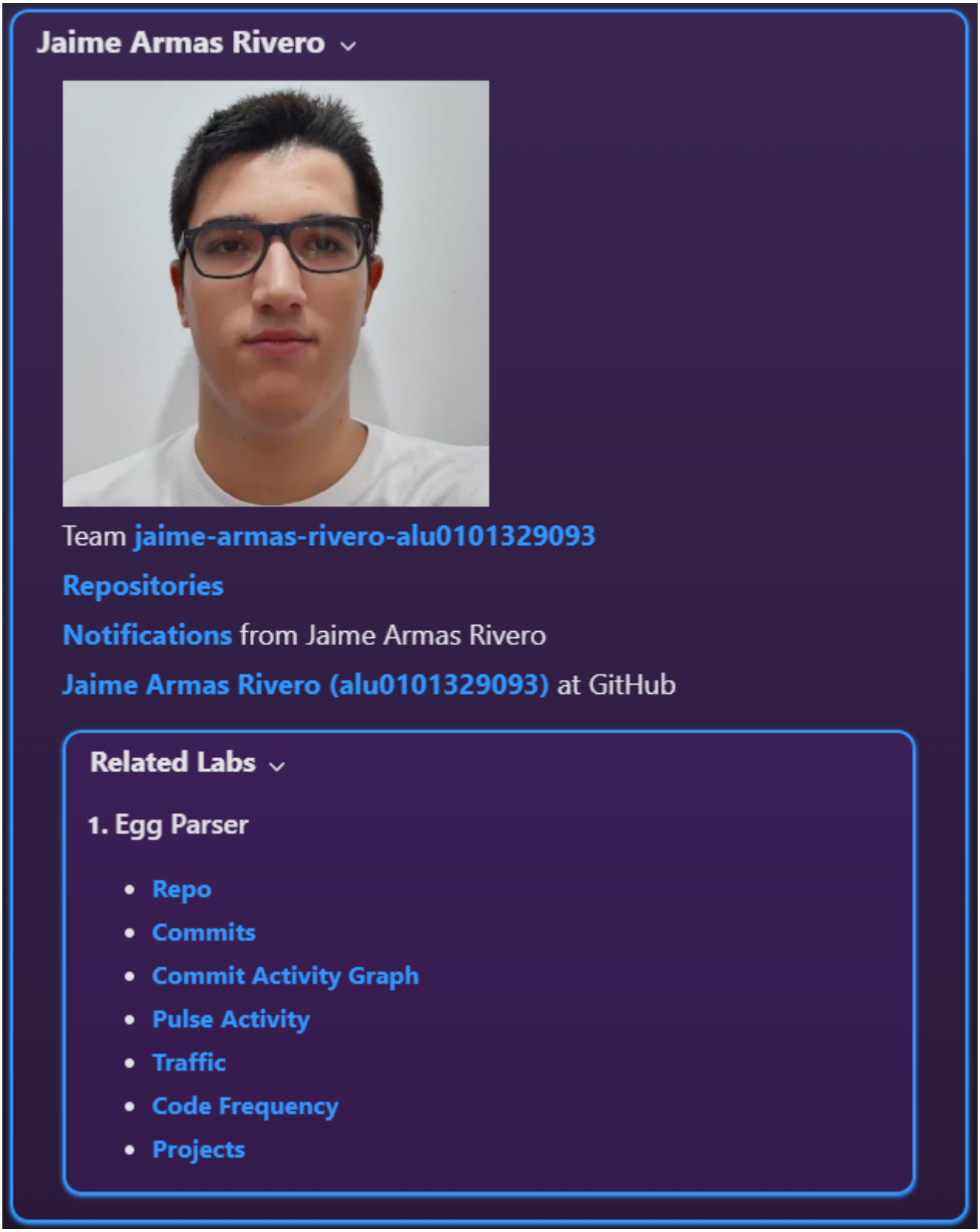

Figura 5.21: Carta con la información de un estudiante

Capítulo 6

Conclusiones y líneas futuras

Lo que más resalto en este trabajo de fin de grado ha sido la tarea de diseñar una aplicación que sea generalista y fácil de usar para el usuario, pero sin perder la potencia y personalización que ofrecería crear un página web desde cero por el usuario. Es por esto, que siempre que se diseñaba nuevo contenido para el LMS nos preguntábamos si no rompía con la filosofía de que la experiencia de usuario siempre sea buena, y en caso de que no fuera así, aunque fuera algo muy potente, se dejaba de lado para siempre pensar en el usuario. Un ejemplo de ello fue el intentar añadir un sistema para autenticarse en el LMS, pero este sistema tenía la pega de que complicaría mucho lo que tendría que configurar el usuario, por esto aunque la idea era buena y estaba casi terminada se dio marcha atrás.

Aunque gran parte del trabajo ha sido cuesta arriba debido a las diferentes dificultades que encontramos por el camino de crear un LMS que fuera lo mejor posible, al final se consiguió crear un sistema que aunque pueda parecer simple a primera vista, siempre se le da la oportunidad al usuario de controlar casi al 100 % el resultado deseado.

Esto junto al hecho de que aprendí un gran numero de tecnologías, de las que antes no sabía nada o muy poco ha hecho que realmente el trabajo y esfuerzo halla valido la pena. Hay veces que incluso nos dejamos llevar por intentar siempre probar cosas nuevas que llego un punto en el que se tuvo que poner un límite, porque aunque la experiencia de aprender algo nuevo es muy bonita, al final este trabajo tenía una fecha final con la que cumplir, aun así las experiencias finales que me llevo me ayudarán en mi futuro y eso lo agradezco. El trabajo de fin de grado está terminado, pero el proyecto puede crecer tanto como desee. Este tipo de proyecto requiere de trabajo constante y prolongado para que tenga acogida en los usuarios. Mis intenciones son seguir en algún momento contribuyendo a este proyecto y tomarlo como un proyecto personal.

Capítulo 7

Summary and Conclusions

"Software Engineering Is Programming Integrated Over Time"

What stands out the most in this final degree project is the task of designing an application that is user-friendly and easy to use, while still maintaining the power and customization that would be offered by creating a webpage from scratch. That's why, whenever we designed new content for the LMS, we questioned whether it would break the philosophy of always providing a good user experience. If it didn't align with that goal, even if it was a powerful feature, we set it aside to prioritize the user. An example of this was attempting to add an authentication system to the LMS, but it would have made the user configuration more complex. So, even though the idea was good and almost finished, we decided to backtrack.

Although a large part of the work has been challenging due to the various difficulties we encountered while creating the LMS to be the best it could be, we managed to create a system that, although it may seem simple at first glance, always gives the user the opportunity to control the desired outcome to nearly 100

This, coupled with the fact that I learned a great number of technologies that I knew nothing or very little about before, has made all the work and effort worthwhile. Sometimes, we get carried away trying to always experiment with new things, but there came a point where we had to set a limit because, despite the joy of learning something new, this project had a set deadline. Nevertheless, the final experiences I gained from it will benefit me in the future, and for that, I am grateful.

The final degree project is completed, but the project can continue to grow as much as desired. This type of project requires constant and prolonged work to gain acceptance among users. My intentions are to continue contributing to this project at some point and treat it as a personal endeavor.

Capítulo 8

Presupuesto

Todas las tecnologías usadas son *open source* y gratuitas y se presupone que el ordenador usado es personal, por lo que no se incluye en el presupuesto
El hipotético saldo son 16 euros la hora

Coste	**Tipos**	**Descripción**
1600	Diseño y planificación	Incluyendo el tiempo invertido en investigación de las nuevas tecnologías necesarias y el diseño y planificación de la aplicación se ha consumido un tiempo sustancial de 100 horas
2720	Desarrollo de los prototipos y aplicación final	Aproximadamente se ha dedicado unas 170 horas
480	Elaboración del informe y conclusiones	Al tratarse de un ecosistema *open source* lanzado a un público general, la documentación es importante, por lo que se ha desarrollado no solo este informe. si no documentación para usuario y desarrolladores en los respectivos repositorios. Necesitando un total de 30 horas

Tabla 8.1: Tabla del presupuesto

Esto supone un presupuesto total de, 4800 euros en un periodo de 8 meses

Apéndice A

Repositorios del ecosistema

Se adjuntan los enlaces a los repositorios de GitHub de los diferentes códigos en los que se ha trabajado.

- EL monorepo con el código del proyecto se encuentra en
  https://github.com/gh-cli-for-education/adastra
- El código de la `memoria`:
  https://github.com/gh-cli-for-education/TFG-2223-Jaime-Armas-Rivero-memoria
- Despliegue de ejemplo con AdAstra:
  https://regular-ring-flax.vercel.app/

Glosario

API Una **interfaz de programación de aplicaciones** es una manera de que dos o más programas informáticos se comuniquen entre sí. Es un tipo de interfaz de software que ofrece un servicio a otras piezas de software. 47

CLI Es un tipo de interfaz de usuario de computadora que permite a los usuarios dar instrucciones a algún programa informático o al sistema operativo por medio de una línea de texto simple. 47

CWD CWD (Current Working Directory) se refiere al directorio actual en el que se está trabajando dentro de un sistema de archivos. Es la ubicación o carpeta en la que un programa o usuario se encuentra actualmente realizando operaciones o ejecutando comandos. 47

framework Un framework es una estructura de software que proporciona herramientas y bibliotecas para facilitar el desarrollo de aplicaciones. Actúa como una base sobre la cual se construyen aplicaciones, ahorrando tiempo al ofrecer soluciones predefinidas para tareas comunes. Los frameworks permiten a los desarrolladores enfocarse en la lógica de su aplicación en lugar de preocuparse por detalles de implementación, lo que agiliza el proceso de desarrollo y promueve la reutilización de código.. 47

JSX Es una extensión de JavaScript que combina HTML y JavaScript para facilitar la creación de interfaces de usuario en aplicaciones web, especialmente en frameworks como React.. 47

MPA Multi-Page Application (MPA) se refiere a un tipo de aplicación web que consta de varias páginas individuales. Cada página representa una vista diferente de la aplicación y se carga por separado del servidor cuando el usuario navega entre ellas. A diferencia de las Single-Page Applications (SPA), que cargan una única página y actualizan su contenido de forma dinámica, las MPAs requieren una carga completa de cada página, lo que puede generar una experiencia más tradicional de navegación.. 47

SSG SSG (Static Site Generation) es un enfoque de desarrollo web donde las páginas web se generan de antemano durante la compilación, lo que resulta en un rendimiento rápido y una menor carga en el servidor.. 47

SSR Server-Side Rendering (SSR) es un enfoque en el desarrollo web donde el servidor genera y envía la página completa al cliente como HTML listo para ser mostrado, ofreciendo una carga inicial más rápida.. 47

Bibliografía

Administrar aulas - documentación de GitHub. (s.f.). Consultado el 13 de julio de 2023, desde https://docs.github.com/es/education/manage-coursework-with-github-classroom/teach-with-github-classroom/manage-classrooms

Astro. (s.f.-a). *Astro Islands*. Consultado el 13 de julio de 2023, desde https://docs.astro.build/en/concepts/islands/

Astro. (s.f.-b). *Astro Syntax*. Consultado el 11 de julio de 2023, desde https://docs.astro.build/en/core-concepts/astro-syntax/

Astro. (s.f.-c). *Content collections*. Consultado el 11 de julio de 2023, desde https://docs.astro.build/en/guides/content-collections/

Astro. (s.f.-d). *Routing*. Consultado el 11 de julio de 2023, desde https://docs.astro.build/en/core-concepts/routing/

Astro. (2023a). *npm: @astrojs/cli-kit* (prog.; Ver. 0.2.3). https://www.npmjs.com/package/@astrojs/cli-kit

Astro. (2023b, mayo). *Astro* (prog.; Ver. 2.4.2). https://astro.build/

Atlassian. (s.f.). *BitBucket | Git solution for teams using Jira* (prog.). https://bitbucket.org/product/

Changesets. (2023, marzo). *GitHub - changesets/changesets: A way to manage your versioning and changelogs with a focus on monorepos* (prog.; Ver. 2.26.1). https://github.com/changesets/changesets

Download | Node.js. (s.f.). Consultado el 13 de julio de 2023, desde https://nodejs.org/en/download

ECMA International. (2023, 8 de julio). *Javascript | MDN* (prog.; Ver. ECMAScript 2021). https://developer.mozilla.org/en-US/docs/Web/JavaScript

Git (prog.; Ver. 2.37.3). (s.f.). https://git-scm.com/

GitHub. (s.f.-a). *Engaged students are the result of using real-world tools - GitHub Education* (prog.). https://education.github.com/

GitHub. (s.f.-b). *GitHub Classroom* (prog.). https://classroom.github.com/

GitHub. (2001, 11 de julio). *Documentación de GraphQL API para GitHub - Documentación de GitHub* (prog.). https://docs.github.com/es/graphql

GitHub. (2022, 28 de noviembre). *GitHub Rest API Documentation - GitHub Docs* (prog.; Ver. 2022-11-28). https://docs.github.com/en/rest?apiVersion=2022-11-28

GitLab Inc. (s.f.). *The DevSecops platform* (prog.). https://about.gitlab.com/

Google. (s.f.). *Classroom management Tools & Resources - Google for Education* (prog.). https://classroom.google.com/

Google Inc. (2023, 13 de junio). *Angular* (prog.; Ver. 16.1.0). https://angular.io/

Gruber, J. (2004, 17 de diciembre). *Daring Fireball: Markdown* (prog.). https://daringfireball.net/projects/markdown/

Installing Node.Js via Package Manager | Node.js. (s.f.). Consultado el 13 de julio de 2023, desde https://nodejs.org/en/download/package-manager

Meta. (2022, 14 de junio). *React* (prog.; Ver. 18.2.0). https://react.dev/
Meta, Google, Expo.dev & Tilde. (2022, 10 de mayo). *Home* (prog.; Ver. 1.22.19). https://yarnpkg.com/
Microsoft. (s.f.). *GitHub: Let's build from here* (prog.). https://github.com/
Microsoft. (2023, 1 de junio). *Javascript with syntax for types.* (prog.; Ver. 5.1.3). https://www.typescriptlang.org/
Moodle - Open-source learning platform | Moodle.org (prog.). (s.f.). https://moodle.org/?lang=en
MPAS vs. SPAS. (s.f.). Consultado el 13 de julio de 2023, desde https://docs.astro.build/en/concepts/mpa-vs-spa/
Netlify. (s.f.). *For fast and secure sites | Jamstack*. Consultado el 13 de julio de 2023, desde https://jamstack.org/
OpenJS Foundation. (2023, 20 de junio). *Node.js* (prog.; Ver. 18.16.1). https://nodejs.org/en
Otander, J. (2023, 9 de febrero). *Markdown for the component era | MDX* (prog.; Ver. 2.3.0). https://mdxjs.com/
Overleaf. (s.f.). *Overleaf, online latex editor* (prog.). https://www.overleaf.com/
Pnpm. (2023a). *Pnpm* (prog.; Ver. 7.26.3). https://pnpm.io/
Pnpm. (2023b). *Workspace | pnpm*. Consultado el 10 de julio de 2023, desde https://pnpm.io/workspaces
Preston-Werner, T. (2023, 20 de enero). *Jekyll • Simple, blog-aware, static sites* (prog.; Ver. 4.3.2). https://jekyllrb.com/
Red Hat. (2023, 3 de marzo). *What is Yaml?* Consultado el 11 de julio de 2023, desde https://www.redhat.com/en/topics/automation/what-is-yaml
Registrar una nueva cuenta GitHub - documentación de GitHub. (s.f.). Consultado el 13 de julio de 2023, desde https://docs.github.com/es/get-started/signing-up-for-github/signing-up-for-a-new-github-account
sindresorhus. (2023). *npm: execa* (prog.; Ver. 7.1.1). https://www.npmjs.com/package/execa
SolidJS. (2023, 30 de marzo). *SolidJS* (prog.; Ver. 1.7.0).
Tailwind Labs. (2023, 25 de abril). *Tailwind CSS - rapidly build modern websites without ever leaving your HTML.* (prog.; Ver. 3.3.2). https://tailwindcss.com/
Theme Configuration - Tailwind CSS. (s.f.). Consultado el 13 de julio de 2023, desde https://tailwindcss.com/docs/theme
unjs. (2022, diciembre). *npm: giget* (prog.; Ver. 1.0.0). https://www.npmjs.com/package/giget
Vercel. (2022). *npm: arg* (prog.; Ver. 5.0.2). https://www.npmjs.com/package/arg
Vercel. (2023). *Turborepo* (prog.; Ver. 1.10.7). https://turbo.build/repo
Vercel Inc. (s.f.). *Vercel: Develop. Preview. Ship. for the best frontend teams* (prog.). https://vercel.com/
The world's fastest framework for building websites (prog.; Ver. 0.113.0). (2023, 5 de junio). https://gohugo.io/
zkochan. (2022). *npm: which-pm-runs* (prog.; Ver. 1.1.0). https://npmjs.com/package/which-pm-runs