



MÓDULO PROYECTO

Ciclo Superior Desarrollo de Aplicaciones Web

Departamento: Informática y Comunicaciones

IES “María Moliner”

Curso: 2017/2018

Grupo: S2I

Proyecto: Cochinillos voladores

Alejandro Yuste Escobar

Email: alejandro.yusesc@educa.jcyl.es

Tutor individual: Enrique Carballo Albarrán

Tutor colectivo: Enrique Carballo Albarrán

Fecha de presentación: 27/05/2024

Índice

Índice	1
Objetivos	3
Cuestiones metodológicas	5
Entorno de trabajo	7
Tecnologías.....	7
Herramientas.....	7
Visión general del sistema	8
Límites del sistema.....	8
Funcionalidades básicas.....	8
Usuarios.....	9
Sistemas operativos compatibles.....	9
Despliegue de la aplicación	9
Plataforma tecnológica.....	9
Instalación de la aplicación.....	9
Planificación	18
Presupuesto	21
Coste desarrollo del código.....	21
Coste software y herramientas.....	21
Coste alojamiento.....	22
Análisis del sistema	22
Diseño del sistema	25
Diseño de la base de datos.....	25
Diseño de la interfaz de usuario.....	29
Diseño de la aplicación.....	33
Implementación	33
Entorno de desarrollo.....	33
Estructura del código.....	47
Pruebas	50
Manual de usuario	51
Manual de instalación	64
Conclusiones y posibles ampliaciones	68
Bibliografía	69
Relación de ficheros en formato digital	70
Glosario	72

Objetivos

Los objetivos eran crear una **aplicación web completa y funcional** para poder dar a **conocer al equipo** con toda la información necesaria para los posibles visitantes de la web que estén interesados en este, además de aportar una manera sencilla y centralizada para el equipo de **mostrar noticias** en el que todo el que desee actualizarse pueda acceder a ellas. También, se quería facilitar la **inscripción al equipo** y su gestión mediante un formulario en el que los usuarios pueden rellenar sus datos y estos serán mandados a los administradores, lo que implica una necesidad de una gestión de correos electrónicos. Adicionalmente, algo de interés era poder acceder a las **estadísticas del equipo** hablando de partidos, competiciones, jugadores, etc. por lo que se realizó una pantalla en el que se pueden observar todas estas y además una gestión en la que se pueden insertar, modificar y eliminar los datos que las componen. Para poder llevar a cabo todas estas tareas la aplicación también cuenta con una gestión de usuarios con la que podemos dar acceso de zonas reservadas de la aplicación a las personas adecuadas.

Con el objetivo de que la aplicación sea **accesible de cualquier parte del mundo** se ha obtenido un servidor virtual privado (VPS) en el que se ha desplegado la web y se ha comprado un dominio para que sea de sencillo acceso y reconocimiento.

Cuestiones metodológicas

En este proyecto he seguido la **metodología ágil**, la cual se basa en poder **entregar versiones funcionales al cliente en cuanto sea posible**. Es por esto mismo que no se han realizado muchas tareas de diseño en la primera parte del proyecto, sino que se intenta comenzar con el desarrollo cuanto antes. Esta metodología es algo muy utilizado en empresas de desarrollo, ya que se ha observado que cuando se llevan a cabo muchas tareas de diseño o preparación a la hora de desarrollar software se gasta mucho tiempo para luego, con lo rápidamente cambiante que es el mundo del desarrollo software, se tenga que volver a realizar por modificaciones del cliente o que este simplemente lo cancele ya que no está viendo ningún resultado funcional al mes de haber empezado el proyecto.

En mi caso he desarrollado un método en base a mi experiencia en las dos empresas en las que he trabajado además de diferentes fuentes de información. En este desarrollo un **primer funcional** en el que se encuentra las funcionalidades requeridas de la aplicación de forma muy general, este se desarrolla en una reunión con el cliente y después se le presentará junto a un presupuesto para que lo pueda aceptar si está de acuerdo con los requerimientos de la aplicación y con sus costes.

Tras esto se realizará un **segundo funcional** en el que se desarrollará más en detalle las funcionalidades, como es el caso de una aplicación web se enumeran las pantallas que deberán ser desarrolladas y cada función que debe tener esta, ej. Pantalla de listado de partidos con una lista en la que se muestran los partidos con los campos: Nombre equipo casa, nombre equipo visitante, goles equipo casa, goles equipo visitante, liga a la que pertenece y fecha en el que tiene lugar. Gracias a este segundo funcional los desarrolladores pueden saber las funcionalidades exactas que tienen que llevar a cabo.

A partir de este momento todas **las tareas realizadas por el equipo serán apuntadas**, no tienen porqué ser organizadas pero si se debe contar las horas y el porqué de estas para que la cabeza del equipo pueda guiar al grupo acordemente. Pienso que la cabeza del equipo no puede ser simplemente un gerente, ya que estos no tienen un punto de vista interno del estado del proyecto, sino que es preferible que la cabeza del equipo sea uno más de este que sencillamente guíe a sus compañeros, pero que se priorice la decisión y organización individual. Esto es porque el mayor

problema con las nuevas metodologías ágiles como es “SCRUM” es la necesidad de una cabeza que organice el proyecto, haciendo que todos los encargados de su desarrollo dependan de este/a en todo momento y dando a lugar a momentos en los que no están desarrollando a la espera de nuevas tareas u órdenes que les sean asignadas. Además, si este no está teniendo parte en el proyecto, existe el problema de las constantes reuniones diarias/semanales para ponerle al día cuando los integrantes del equipo ya tienen claro el estado del proyecto y cuales son los siguientes pasos a seguir.

Antes de empezar a desarrollar se realiza un **diseño inicial** para poder observar cómo va a ser el estilo general de la aplicación, este no tiene por qué ser de todas las pantallas ya que, al estar utilizando el método ágil, toda tarea que no sea desarrollo se tomará como pérdida de tiempo, se diseñará solamente lo necesario para que los desarrolladores comprendan la vista general.

Mientras el diseño de la interfaz de usuario es realizado se llevará a cabo el **diseño y creación de la base de datos**, esta es muy probable que sufra cambios a lo largo del desarrollo pero se necesita una primera base para poder comenzar con este. Es recomendable generar o realizar un esquema de la base de datos para que los desarrolladores puedan ver su estructura y desarrollar contando con esta, ya que todo lo que ayude en el desarrollo de la aplicación es una buena inversión. La base de datos es el cimiento de la aplicación así que esta debe ser desarrollada por alguien con experiencia y metodológico, tomándose el tiempo que necesite ya que sin esta o con una base de datos mal diseñada todo el proyecto sufre.

Tras estos dos últimos pasos ya tenemos lo suficiente para realizar una **primera versión de la aplicación**, esta puede ser una pequeña parte del objetivo final ya que como se ha mencionado antes la idea es presentar al cliente con productos funcionales cuanto antes, así podremos recibir críticas o cambios en una fase inicial del desarrollo y será sencillo realizar lo necesario para cumplir las modificaciones.

Para poder presentar al cliente esta primera versión de la aplicación deberemos realizar las **tareas de CI/CD o implementación de código**, donde crearemos un lugar centralizado en el que guardar el código de la aplicación y un servidor para ejecutar esta. Para poder realizar la conexión entre el desarrollador, el repositorio y el servidor de forma sencilla y sin fricción se realizará las tareas que hemos mencionado antes,

con tareas que se ejecutarán automáticamente al subir los cambios al repositorio compilando el código, probándolo y subiéndose al servidor para que el cliente pueda acceder a este.

Después de esta primera pequeña versión para poder realizar la configuración del repositorio y servidor se seguirá con el **desarrollo haciendo pequeñas partes de la aplicación** que estén totalmente finalizadas para que el cliente sea capaz de tener una vista del progreso del proyecto. Esto, como se ha mencionado antes, nos deja adaptarnos a las modificaciones o problemas que pueda ver el cliente en las secciones funcionales de la aplicación, teniendo así una retroalimentación instantánea con la que podemos trabajar sin tener que volver a generar interminables documentos de diseño volviendo al comienzo del proyecto. Es también cierto que, dependiendo de estas modificaciones, se le podrá cobrar al cliente extra ya que se toma el presupuesto como un número provisional y no final; si la modificación se puede llegar a un acuerdo de que puede haber estado reflejada al comienzo o sea error de desarrollo no se aumentará el cobro, pero si el cambio es sustancial y añade trabajo extra a los desarrolladores si que lo hará.

Entorno de trabajo

Tecnologías

En esta aplicación web se está utilizando un modelo centralizado, ya que todas las peticiones irán al VPS que se ha contratado

Herramientas

Las herramientas que se han utilizado en la aplicación son las siguientes:

- HTML
- CSS
- JavaScript
- HTMX
- Rust
- Axum
- Askama
- MySQL

- Sqlx
- Docker
- Github
- Caddy
- Editor: Neovim
- S.O: NixOS

Visión general del sistema

Límites del sistema

Esta es una **aplicación autónoma**, ya que no depende de ninguna API o servicio para insertar los datos necesarios. Estos serán insertados por los administradores para ir actualizando la información como las noticias o los datos de los partidos.

Funcionalidades básicas

Las funcionalidades básicas de la aplicación es en un principio dar a conocer el equipo, esto es mediante la página de inicio. Además, se podrá avisar a los clientes de posibles eventos o sencillamente información que les interese mostrar mediante las noticias, esto implica una **gestión de las noticias** para que los administradores puedan añadir, modificar y eliminarlas. El hecho de tener administradores implica que necesitamos una **gestión de usuarios**, los visitantes de la página podrán registrarse e iniciar sesión, en caso de tener los permisos de administrador entonces podrá acceder a la **gestión de datos** de toda la aplicación. Adicionalmente, la aplicación cuenta con un formulario en la que se les da la oportunidad a los usuarios a apuntarse al equipo, mandando los datos de este formulario a los administradores de la aplicación, esto significa que se necesitará una **gestión de correo** con la que mandaremos la información de interés. Por último, se contará con un apartado en el que los visitantes podrán observar las estadísticas del equipo en los últimos partidos, es por esto que se contará con la **gestión de todos los datos relacionados con las estadísticas**.

Usuarios

En la aplicación va a existir dos tipos de usuarios, los **usuarios generales**, que la única diferencia con los visitantes sin inicio de sesión es que podrán rellenar el formulario de inscripción, y los **usuarios administradores**, que podrán acceder al panel de control en el que podrán gestionar todos los datos de la aplicación, además de que recibirán correos con la información de todo usuario que se apunte al equipo mediante el formulario de inscripción.

Sistemas operativos compatibles

La aplicación en un principio está diseñada para ejecutarse en un **sistema basado en Linux**, en cambio, gracias a estar desarrollada con la herramienta “Docker”, **se podría utilizar en cualquier sistema operativo** ya que esta herramienta nos permite ejecutar el código en un sistema predefinido por su configuración haciendo una simulación de un sistema operativo dentro de la máquina que lo esté ejecutando, creando así una capa de abstracción.

Despliegue de la aplicación

Plataforma tecnológica

En el caso de este proyecto se ha utilizado un VPS con el sistema operativo **Ubuntu 24.04 (LTS) x64**, en cambio dentro de la virtualización de “Docker” se ha utilizado el sistema operativo **Debian Bullseye**.

Instalación de la aplicación

Gracias a utilizar la herramienta “Docker” la instalación de la aplicación es mínima, y se puede realizar como se desee. En el caso del proyecto se ha utilizado la herramienta “Github actions” para poder hacer el CI/CD de la aplicación y DigitalOcean para hospedarla en un VPS. Esto se ha realizado de la siguiente manera:

Lo primeramente necesario es el **servidor virtual privado (VPS) en DigitalOcean**, donde cuentan con grandes opciones que se ajustan a cualquier caso, en el de esta aplicación se utilizó la opción básica de 6 €/mes que cuenta con 1GB de memoria RAM, 1 vCPU, 1.000GB de transferencia y 25GB de almacenamiento.

Basic Droplets

Basic Droplets have the most efficient CPU usage at a lower cost for workloads that underuse dedicated threads. They're ideal for bursty applications that can handle variable levels of CPU.

CPU Options **Regular** Premium Intel Premium AMD

Memory	vCPU	Transfer	SSD	\$/hr	\$/mo	
512 MiB	1 vCPU	500 GiB	10 GiB	\$0.00595	\$4.00	→
1 GiB	1 vCPU	1,000 GiB	25 GiB	\$0.00893	\$6.00	→
2 GiB	1 vCPU	2,000 GiB	50 GiB	\$0.01786	\$12.00	→
2 GiB	2 vCPUs	3,000 GiB	60 GiB	\$0.02679	\$18.00	→
4 GiB	2 vCPUs	4,000 GiB	80 GiB	\$0.03571	\$24.00	→
8 GiB	4 vCPUs	5,000 GiB	160 GiB	\$0.07143	\$48.00	→
16 GiB	8 vCPUs	6,000 GiB	320 GiB	\$0.14286	\$96.00	→

En la configuración del droplet deberemos seleccionar la región más cercana a nuestras preferencias, en el caso de nuestra aplicación está hospedada en Frankfurt, además seleccionaremos el sistema operativo que deseemos en nuestro servidor, en esta configuración se utiliza Ubuntu 24.04 (LTS) x64, finalmente se inserta una contraseña para el usuario root del servidor necesaria más adelante en la configuración de la aplicación y se crea un nuevo proyecto donde guardar este droplet.

cochinillos-voladores

Image

Size

Ubuntu 24.04 (LTS) x64

1 vCPU
1GB / 25GB Disk
(\$6/mo)
[Resize](#)

Region

IPv4

IPv6

Private IP

VPC

FRA1

[Enable](#)

[default-tra1](#)

Ahora que tenemos reservado el servidor virtual privado (VPS) podremos acceder a su consola mediante ssh o la consola que nos proporciona DigitalOcean dentro del panel de control del droplet. Una vez dentro deberemos **instalar los paquetes necesarios** para correr el código mediante el gestor de paquetes del sistema operativo elegido, en el caso de Ubuntu se deberá actualizar el sistema, instalar docker y el plugin de docker compose.

```
sudo apt-get update
sudo apt-get install docker.io
mkdir -p ~/.docker/cli-plugins/
curl -SL https://github.com/docker/compose/releases/download/v2.3.3/docker-compose-linux-x86_64 \
-o ~/.docker/cli-plugins/docker-compose
```

Una vez realizados estos pasos se crea un repositorio en Github donde se sube el código de la aplicación de forma manual o mediante comandos, en este repositorio se insertan las variables de entorno necesarias para ejecutar el código de manera anónima. Estos se pueden configurar en los ajustes del repositorio y deben ser los siguientes:

- **[Secreto] CONTRA_MAIL.** Contraseña del usuario que tengamos en el proveedor de mail.
- **[Secreto] USUARIO_MAIL.** Usuario del proveedor de mail
- **[Secreto] DATABASE_TEST_URL.** Contraseña para la base de datos que se crea para las pruebas en la implementación
- **[Secreto] DOCKER_PASSWORD.** Contraseña de nuestra cuenta en Docker Hub
- **[Secreto] DOCKER_USERNAME.** Usuario de Docker Hub
- **[Secreto] DROPLET_PASSWORD.** Contraseña de nuestro droplet en DigitalOcean
- **[Secreto] MYSQL_PASSWORD.** Contraseña de la base de datos que se creará en producción
- **[Variable] DROPLET_ID.** Id del droplet que tengamos contratado en DigitalOcean
- **[Variable] MAIL.** Dirección de correo que hayamos contratado en el proveedor de mail

- **[Variable] RUTA_UPLOADS.** Ruta que se utilizará para subir los archivos en la aplicación cuando se ejecute en producción

Para poder determinar los procesos de integración en github mediante github actions se crea un archivo con extensión yml dentro de una carpeta “workflows” en otra llamada “.github”. En el caso de esta aplicación se ha creado un archivo “prod.yml” para compilar, probar y subir el código cada vez que se haga un cambio en la rama “main”, que es la rama principal de este proyecto.

En este archivo podremos ver una primera configuración con la que diremos a github como se llama el proceso y cuando se debe ejecutar, en este caso, cuando se haga push a la rama main:

```
name: Compilar, probar y subir a producción

on:
  push:
    branches:
      - main
```

Después configuraremos las variables de entorno necesarias para la implementación, SQLX_OFFLINE, con la que especificaremos a “cargo build” que no queremos que intente acceder a la base de datos especificada en DATABASE_URL y DATABASE_URL, con la que se especifica la cadena de conexión con la base de datos de prueba contra la que podremos ejecutar “cargo test”, la cual hemos especificado anteriormente en los secretos:

```
env:
  SQLX_OFFLINE: true
  DATABASE_URL: ${ secrets.DATABASE_TEST_URL }
```

Ahora dividiremos el proceso en dos trabajos, uno para compilar el código (build) y probarlo y un segundo para subir el código a nuestro VPS (deploy), ambos correrán en “ubuntu-latest” y el trabajo “deploy” necesitará que se haya ejecutado correctamente el trabajo “deploy”:

```
jobs:
  build:
    runs-on: ubuntu-latest
    // - corte -
  deploy:
    needs: build
    runs-on: ubuntu-latest
```

En el trabajo de “build” comenzaremos con especificar los servicios que queremos que corran durante este, en este caso un servidor mysql donde insertamos más adelante la estructura de base de datos para poder probar el código mediante “cargo test”, utilizaremos la imagen de mysql la cual configuraremos con dos variables de entorno, MYSQL_DATABASE siendo la base de datos que creará por defecto y MYSQL_ROOT_PASSWORD la contraseña del usuario root con el que accederemos:

```
services:
  mysql:
    image: mysql:latest
    env:
      MYSQL_DATABASE: cochinillos
      MYSQL_ROOT_PASSWORD: ${ secrets.MYSQL_PASSWORD }
    ports:
      - 3306:3306
```

Tras esto configuraremos los pasos que llevará a cabo el trabajo dentro de la lista “steps”, comenzando con descargar el código de nuestro repositorio, en este caso como el código está bajo la ruta “codigo/cochinillos-voladores” hay que especificar que sea esta la que copie además del “sparse-checkout-cone-mode” para que esta sea la única que copie, además de la ruta donde queremos que la copie que es la inicial:

```
steps:
- name: Checkout codigo
  uses: actions/checkout@v4
  with:
    path: ./
    sparse-checkout: |
      codigo/cochinillos-voladores
    sparse-checkout-cone-mode: false
```

Aún habiendo especificado la carpeta que queremos copiada deberemos mover los archivos del código fuera de esta con el siguiente paso, utilizaremos comandos de linux para este paso ya que estamos corriendo el trabajo en Ubuntu:

```
- name: Mover archivos
  run: |
    ls -lah
    mv codigo/cochinillos-voladores/* ./
    rm -rf codigo
    ls -lah
```

Después instalaremos las herramientas de rust para poder compilar y probar el código:

```
- name: Instalar rust
  uses: actions-rs/toolchain@v1
  with:
    profile: minimal
    toolchain: stable
```

Además, instalaremos la herramienta “sqlx-cli” para manejar las migraciones y generaciones relacionadas con la base de datos:

```
- name: Install SQLx CLI
  run: cargo install sqlx-cli --no-default-features --features native-tls,mysql
```

Ejecutaremos las migraciones mediante la herramienta que acabamos de instalar para popular la base de datos de pruebas con la estructura y datos necesarios:

```
- name: Run SQLx Database Migrations
  run: sqlx migrate run
```

Generamos los archivos necesarios para poder ejecutar el código correctamente con la misma herramienta:

```
- name: Generate SQLX Prepared Queries
  run: cargo sqlx prepare
```

Es entonces cuando podremos compilar y probar el código mediante la herramienta que nos ofrece rust llamada “cargo”:

```
- name: Compilar y probar código
  run: |
    cargo build --verbose
    cargo test --verbose
```

Ahora configuraremos el constructor de docker mediante una nueva acción:

```
- name: Configurar Docker Buildx
  uses: docker/setup-buildx-action@v3
```

Iniciaremos sesión en nuestra cuenta de docker hub con la acción que nos ofrece docker, utilizaremos los secretos que hemos configurado anteriormente en el repositorio:

```
- name: Iniciar sesión en docker
  uses: docker/login-action@v3
  with:
    username: ${ secrets.DOCKER_USERNAME }
    password: ${ secrets.DOCKER_PASSWORD }
```

Ahora subiremos la imagen que generemos a partir del “Dockerfile” a docker hub:

```
- name: Compilar y subir imagen docker
  uses: docker/bake-action@v4
  with:
    push: true
```

El siguiente trabajo es el de “deploy” en el que comenzaremos con obtener el código del repositorio como anteriormente y lo moveremos también:

```
- name: Checkout codigo
  uses: actions/checkout@v4
  with:
    path: ./
    sparse-checkout: |
      codigo/cochinillos-voladores
    sparse-checkout-cone-mode: false

- name: Mover archivos
  run: |
    ls -lah
    mv codigo/cochinillos-voladores/* ./
    rm -rf codigo
    ls -lah
```


Volveremos a iniciar sesión en docker hub como lo hicimos anteriormente:

```
- name: Iniciar sesión en docker
  uses: docker/login-action@v3
  with:
    username: ${ secrets.DOCKER_USERNAME }
    password: ${ secrets.DOCKER_PASSWORD }
```

Ahora instalaremos “sshpass” para poder copiar los archivos necesarios para correr la aplicación a nuestro servidor virtual privado:

```
- name: Instalar sshpass
  run: sudo apt-get install sshpass
```

Con esta herramienta copiaremos los archivos “Caddyfile” en el que tenemos la configuración de la redirección y así poder conseguir https, “docker-compose.yml” en el que tenemos la configuración de todos los contenedores necesarios en la ejecución de la aplicación y “docker-compose-prod.yml” en el que tenemos la configuración del contenedor necesario para poder ejecutar el servidor de redirección caddy.

```
- name: Copiar Caddyfile
  run: |
    sshpass -v -p ${ secrets.DROPLET_PASSWORD } \
    scp -o StrictHostKeyChecking=no \
    Caddyfile root@${ vars.DROPLET_IP }:~

- name: Copiar docker-compose.yml
  run: |
    sshpass -v -p ${ secrets.DROPLET_PASSWORD } \
    scp -o StrictHostKeyChecking=no \
    docker-compose.yml root@${ vars.DROPLET_IP }:~

- name: Copiar docker-compose.prod.yml
  run: |
    sshpass -v -p ${ secrets.DROPLET_PASSWORD } \
    scp -o StrictHostKeyChecking=no \
    docker-compose.prod.yml root@${ vars.DROPLET_IP }:~
```

Por último, utilizaremos la acción “appleboy/ssh-action” para poder ejecutar comandos en nuestro VPS de forma que reiniciaremos el código que está corriendo en nuestro servidor, o lo que es lo mismo, los contenedores de docker:

```
- name: Deploy
  uses: appleboy/ssh-action@v1.0.3
  with:
    host: ${vars.DROPLET_IP}
    username: root
    password: ${secrets.DROPLET_PASSWORD}
    script: |
      cd ~
      export MYSQL_PASSWORD=${secrets.MYSQL_PASSWORD}
      export RUTA_UPLOADS=${vars.RUTA_UPLOADS}
      export MAIL=${vars.MAIL}
      export USUARIO_MAIL=${secrets.USUARIO_MAIL}
      export CONTRA_MAIL=${secrets.CONTRA_MAIL}
      docker compose down
      docker compose pull
      docker compose -f docker-compose.yml -f docker-compose.prod.yml up -d
```

Ahora que tenemos esto configurado ya podremos hacer push en nuestro repositorio y este ejecutará el proceso para subir los cambios en la web.

Planificación

La planificación se llevó a cabo con la idea de realizar el desarrollo mediante el método ágil anteriormente explicado, es por esto que la inicial se dividió en las siguientes actividades:

Tareas iniciales:

- Creación del primer funcional [1-2 días]
- Creación del segundo funcional [2-3 días]
- Creación del repositorio [1 día]

Diseño:

- Diseño de la base de datos [2-5 días]
- Diseño de la interfaz de usuario [5-8 días]

Desarrollo:

- Código inicial [5-8 días]
- Desarrollo del CI/CD [2-3 días]
- Desarrollo de la aplicación [3-4 semanas]

Documentación:

- Creación de la memoria del proyecto [1-2 semanas]

Sin embargo, los tiempos reales son esperadamente diferentes. Empezando con las tareas iniciales, se asemeja en gran parte a los tiempos esperados ya que son tareas mundanas y sencillas, sencillamente hay que llegar a un acuerdo con los objetivos de la aplicación y generar un documento a partir de ello.

En el caso del diseño de la base de datos entra dentro de lo esperado llevando simplemente 2 días, sin embargo sí que hubo algunos cambios durante el desarrollo con lo que podríamos añadir 2 días más. Además, es de notar que la gran mayoría de tiempo en el diseño de la base de datos se pasó generando documentación o diagramas en vez de diseñando y creando esta, es por eso que se debería intentar encontrar mejores métodos, herramientas o simplemente acortar el número de documentación y centrarse en el desarrollo inicial de la base de datos.

El diseño de la interfaz de usuarios también entra dentro de la idea inicial llevando 6 días de trabajo, esto en gran parte es gracias a la metodología que se llevó a cabo de no tener que terminar todo el diseño de la interfaz, ya que una vez en el desarrollo, con la idea general que daban los elementos más importantes que fueron los que se diseñaron, no hizo falta ningún previo diseño de una pantalla para poder saber cómo llevarla a cabo para que concuerde con el resto de la aplicación.

El código inicial si llevó más tiempo de lo esperado (9 días), siendo esto por la inexperiencia de inicio de proyectos que tenía ya que nunca he tenido la oportunidad de desarrollar uno desde cero. Es por esto que la grán mayoría de tiempo se perdió en la búsqueda de plantillas y estructuras generales en las que otras personas hubiesen desarrollado proyectos anteriormente, el problema siendo que la tecnología escogida no es una muy utilizada y por consiguiente no existía gran cantidad de información o ejemplos. Sin embargo, después de invertir tanto tiempo en el comienzo del proyecto, el desarrollo fué infinitamente más sencillo gracias a tener una base tan robusta, metódica y organizada, es por esto que no dejaría jamás de recomendar que siempre se tome una gran cantidad de tiempo en el inicio y preparación de una aplicación para que no existan fricciones en el desarrollo de esta y no sea necesario volver a la base pudiendo romper toda la aplicación con una simple modificación. En mis próximos proyectos tendré mejor en cuenta que este inicio lleve tanto tiempo.

El desarrollo de CI/CD también excedió los tiempos, el inicio solamente llevó 3 días siendo la mayoría de este tiempo arreglando y buscando información sobre las diferentes herramientas que se han utilizado, sin embargo, estos días fueron para subir el código y configurar el servidor contratado, con lo cual debemos sumar los días que se tuvo que modificar la configuración del CI/CD para crear un almacenamiento donde poder mantener los archivos que suban los usuarios de la aplicación (1 día) y la configuración del servicio de correo (1 día). Esto en gran parte es por la misma razón que el anterior punto, al ser tecnologías tan poco utilizadas no hay mucha documentación para como conectarla y manejarla a la hora de la implementación, aunque otra gran parte es la inexperiencia que tenía con herramientas de CI/CD y la configuración de servidores para estas.

Por último, el desarrollo también entró dentro del rango predefinido (13 días) ya que, como se ha mencionado antes, en el inicio se hicieron los preparativos suficientes para llevar a cabo un desarrollo eficiente y de calidad, es por esto que, exceptuando cambios que necesarios como la configuración del servidor de correos y el manejo de subida de archivos de parte de los usuarios, no hubo ningún inconveniente que frenase la entrega de secciones de la aplicación funcionales.

Presupuesto

Coste desarrollo del código

A la hora de hacer el presupuesto, el coste del desarrollo, según me he informado con las empresas en las que he trabajado, he decidido realizarlo con la experiencia de otros proyectos, esto sumado a poner un precio de 20€/hora nos da un presupuesto medianamente real de lo que costaría el desarrollo de cada aplicación, en el caso de esta aplicación, sumando todos los peores casos de la anterior predicción de tiempos, contando con las semanas siendo de 5 días, resultan unos 50 días, a los que podremos adjudicar 4 horas al día siendo un total de 200 horas y por último un presupuesto de 4.000€. Este sería un presupuesto inicial de solamente el desarrollo del código y al que se sumaría los demás costes, entregándolo al cliente junto al primer funcional.

Sin embargo, el coste real es diferente, si observamos los tiempos reales suman un total de 41 días, lo que llevaría a 164 horas y por último unos 3.280€. También es cierto que el presupuesto es un precio cerrado que no bajará ni subirá una vez terminada la aplicación, siendo algo justo ya que puede costar menos y hacer que la empresa desarrolladora gane dinero o que cueste más y que lo pierda.

Coste software y herramientas

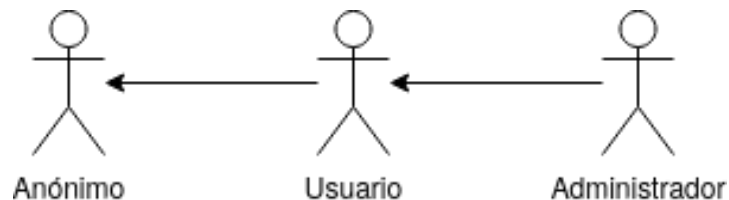
Gracias a que se ha priorizado el uso de herramientas de código abierto las herramientas que se han utilizado no han resultado en coste alguno, sin embargo si podemos contar con costes como el dominio y el servicio de correo electrónico. Ambos han sido contratados en la misma compañía, Porkbun, costando respectivamente 10.72\$/año por el dominio y 24\$/año por el servicio de correo. Elegí esta compañía ya que el precio por ambos servicios es bastante competitivo y además ofrecen considerables ventajas que en otras compañías te cobran, como es la anonimidad del comprador en el dominio, y por un gran precio, sumando a todo esto su gran atención al cliente por cualquier problema o cuestión que puedas tener.

Coste alojamiento

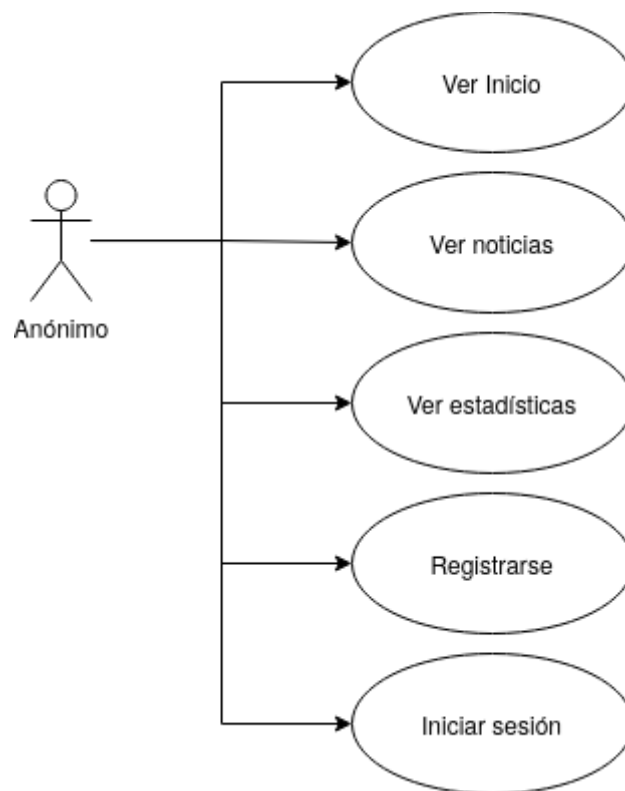
Gracias a utilizar un servidor virtual privado los costes del alojamiento son mínimos, ya que cualquier otro servicio te cobraría por el adicional mantenimiento que en nuestro caso estamos haciendo nosotros, no obstante, gracias a las herramientas utilizadas, este mantenimiento es cerca de nulo. Este precio varía en la capacidad de servidor que escojamos, pero en el caso de esta aplicación se escogió la opción de 6€/mes, cifra extremadamente competitiva contando con los precios de otras compañías diferentes a DigitalOcean y añadiendo la increíble calidad de las herramientas que te proporcionan y la gran cantidad de documentación en infinidad de temas que esta empresa ofrece de forma gratuita y sencilla a todo el que necesite sin necesidad de tener contratados sus servicios.

Análisis del sistema

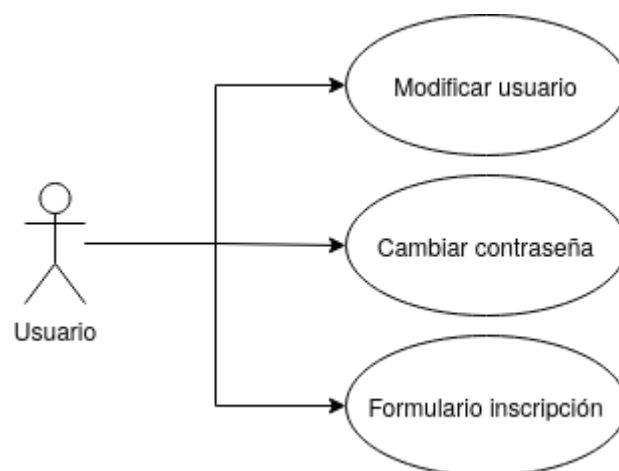
En esta aplicación, como se ha mencionado anteriormente, existen dos tipos de usuarios, sin contar el visitante no iniciado, estos ganan permisos manteniendo los de su anterior estado, el usuario tiene los permisos de un anónimo añadidos a los suyos, el administrador tiene los permisos de ambos añadidos a los suyos. Esto se puede observar en el siguiente gráfico:



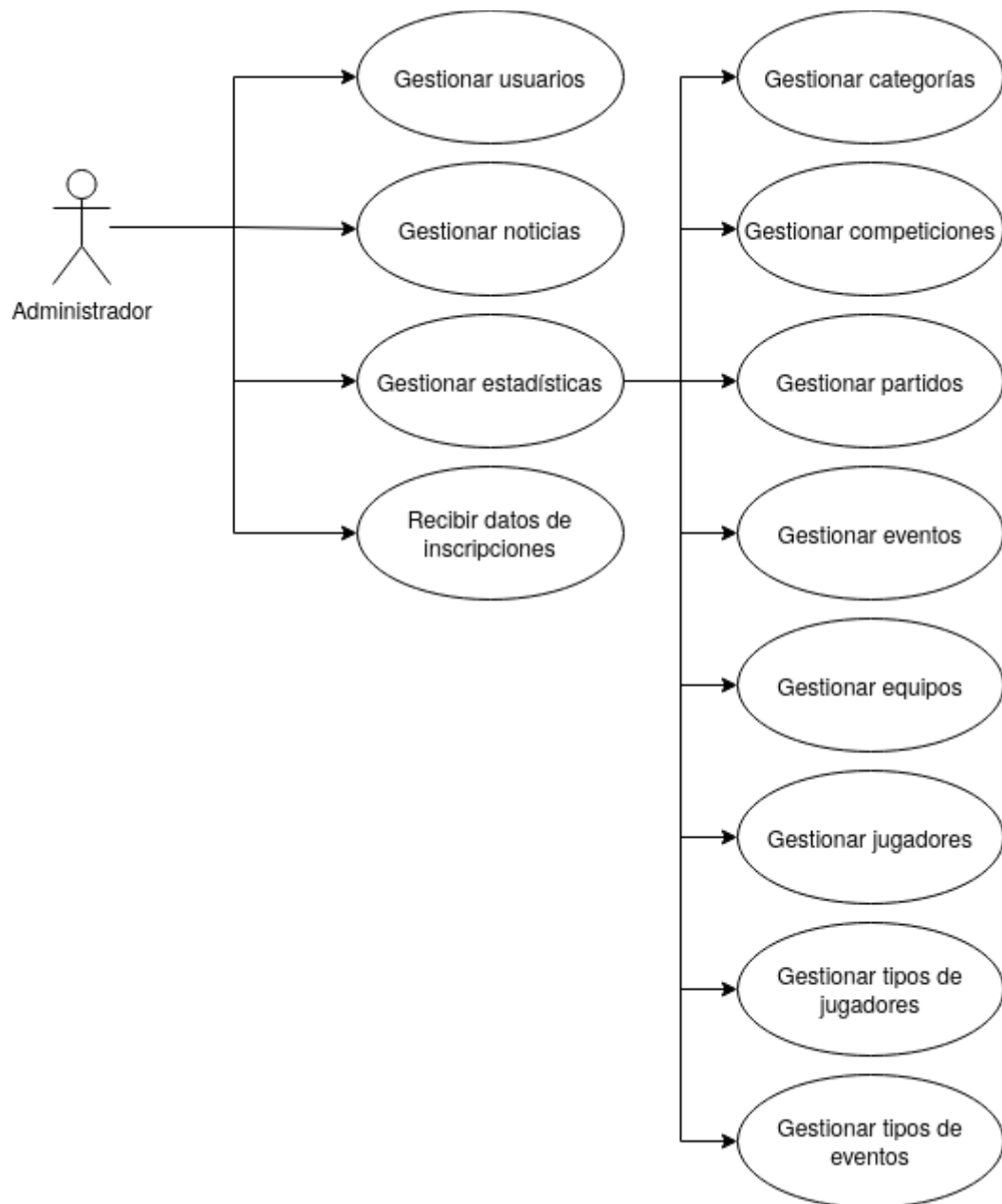
Ahora, los permisos que tiene el actor anónimo son los básicos de la aplicación, como observar la información del inicio, la lista de las noticias, y las estadísticas de los partidos, además de tener la opción de registrarse o iniciar sesión para obtener los permisos de usuario:



Después, podemos añadir los casos de uso del usuario, siendo estos el poder modificar sus datos, cambiar su contraseña y apuntarse al equipo mediante el formulario de inscripción:



Por último, el administrador cuenta con todos los permisos de la aplicación, siendo estos la gestión de usuarios, exceptuando las contraseñas, las noticias y todos los datos relacionados con las estadísticas o lo que es lo mismo los partidos para que todos estos datos se muestren en la aplicación de manera correcta según los deseos de los administradores, además de recibir los datos de las inscripciones:

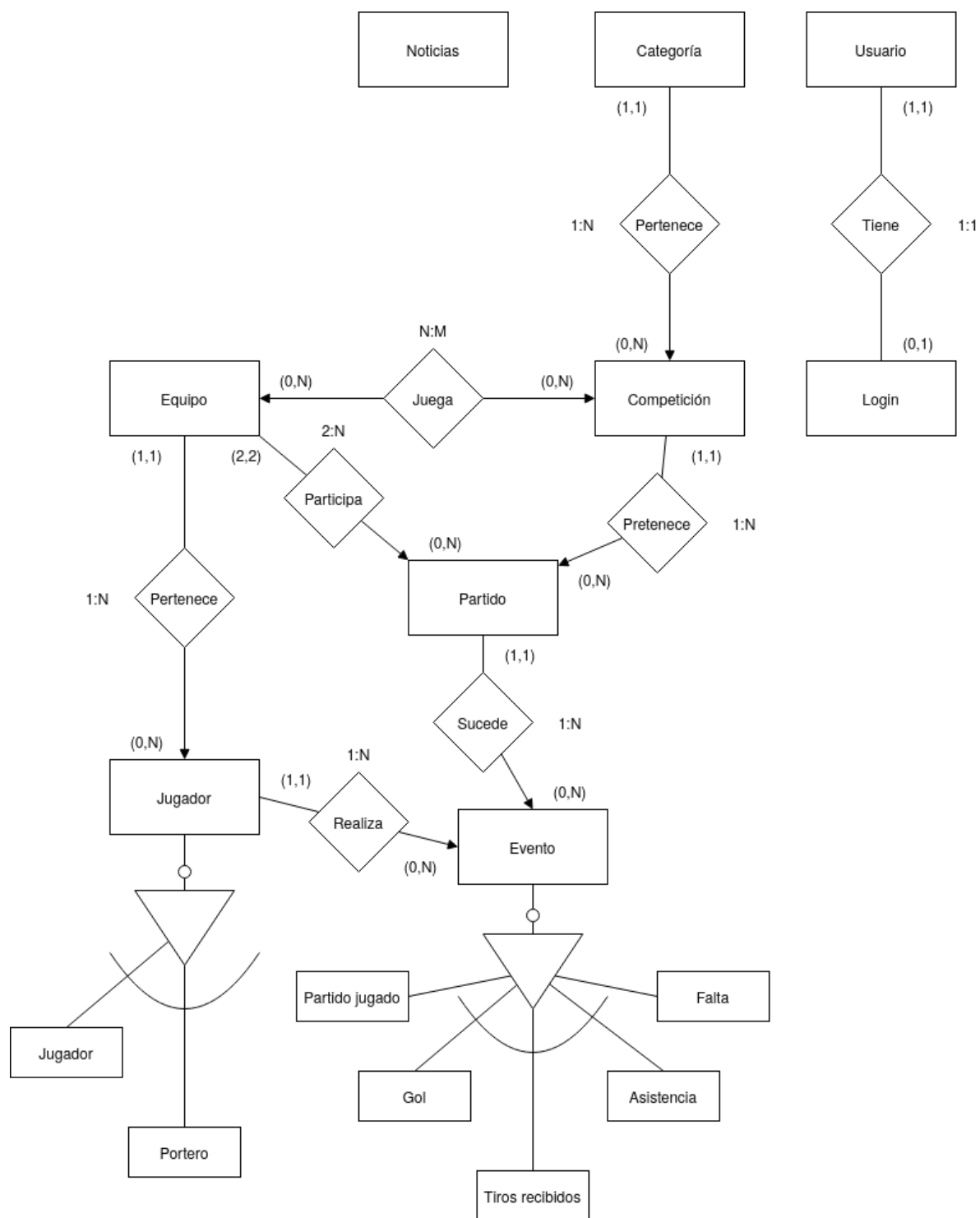


Diseño del sistema

Diseño de la base de datos

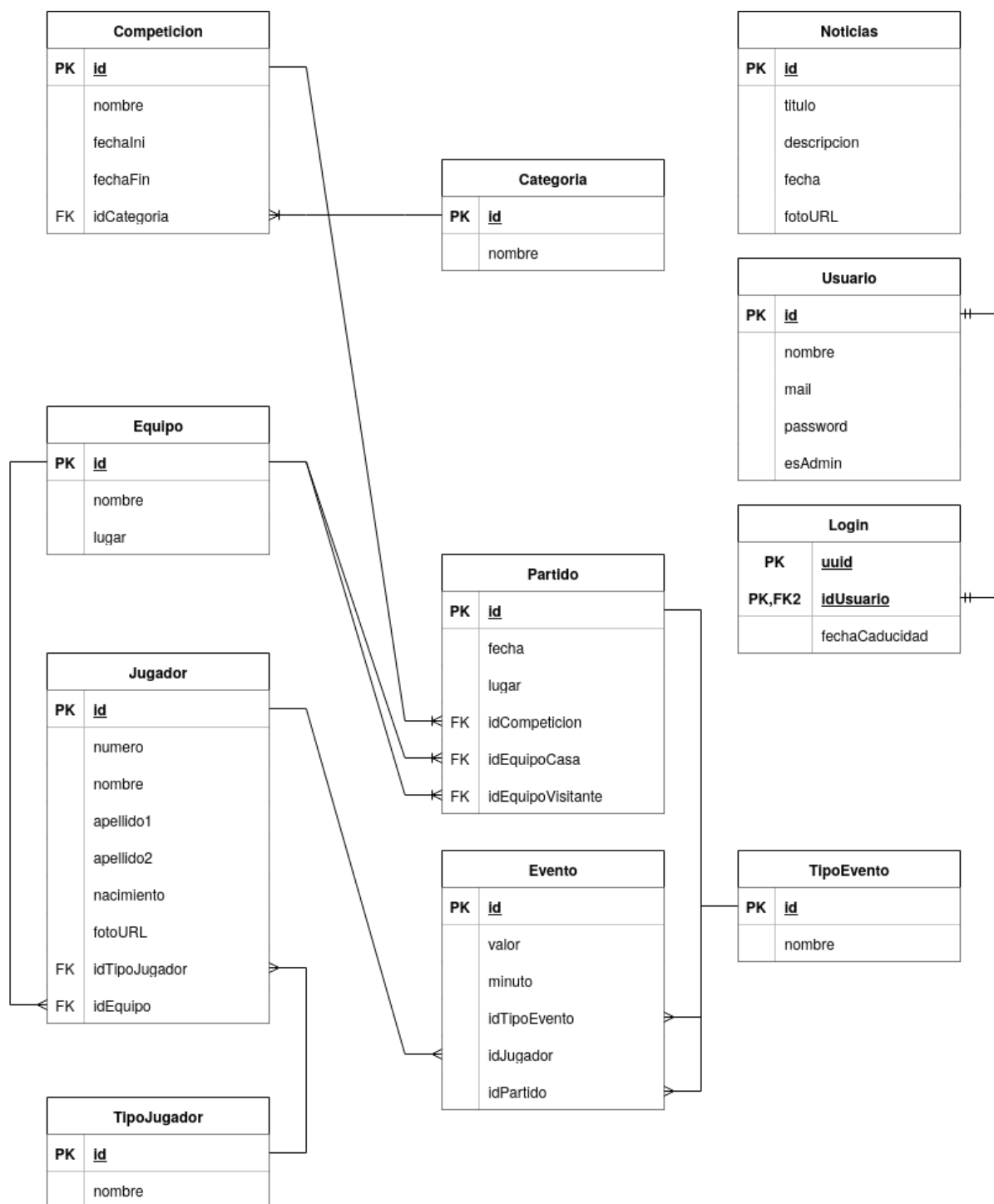
Se ha desarrollado un modelo E/R para observar las relaciones de manera sencilla entre todas las entidades. Empezando con las más sencillas, la entidad "**Noticias**" no tiene relación ninguna ya que esta solo representa las noticias que aparecerán en el blog de la aplicación, así mismo, la entidad "**Usuario**" solamente tiene relación con la tabla de "**Login**" ya que esta es la que relaciona los UUID que guardan las cookies de los diferentes navegadores en los que se haya iniciado un usuario con el mismo usuario, pudiendo así autenticar a este.

Después, las entidades más relacionadas empiezan por "**Competición**", que representa las competiciones que se llevarán a cabo separadas por "**Categorías**". Dentro de estos "**Equipos**" se encuentran los "**Jugadores**" de la plantilla. Y para conseguir las estadísticas se han creado los "**Partidos**" que se llevarán a cabo en las "**Competiciones**" entre dos "**Equipos**", y dentro de estos partidos ocurrirán diferentes "**Eventos**" que estarán relacionados con ciertos "**Jugadores**" y "**Partidos**".



El modelo relacional muestra las siguientes tablas:

- **Usuario.** Lo único que necesitamos es su nombre para mostrarlo, el email al que enviar los correos y si es administrador para que pueda subir entradas al blog.
- **Logins.** Tabla utilizada para la autenticación de los usuarios mediante una relación entre un UUID que se guardará en una cookie del buscador y el usuario al que se quiere acceder, caducando esta cuando se supere la fecha.
- **Noticias.** En este insertamos todas las entradas que dispondrán de un título, una descripción, una imagen y la fecha en la que se subió.
- **Competición.** Guardaremos el nombre que se mostrará de esta, las fechas entre las que se celebra y a qué categoría pertenece.
- **CompeticionEquipo.** Utilizaremos esta tabla para relacionar a todos los equipos con las competiciones que se llevan a cabo.
- **Equipo.** En la que guardaremos su nombre, el lugar al que pertenece y su categoría.
- **Categoría.** Simplemente se guarda el nombre que se mostrará en la página.
- **Partido.** Se guardará la fecha en el que se discutió, el lugar, a que competición pertenece y los dos equipos que jugaron entre ellos.
- **Jugador.** Necesitamos su número, nombre, los dos apellidos, su fecha de nacimiento para poder mostrar su edad, una foto de este y al equipo al que pertenece.
- **TipoJugador.** Este es para poder saber la posición que ocupa el jugador, portero o jugador.
- **Evento.** Estas serán acciones que pueden suceder en un partido, serán de un tipo de la tabla de tipos, estará relacionado con un jugador y guardaremos el partido en el que ha sucedido. Además, dependiendo del tipo se usará el minuto y el valor de diferente manera, en "partido jugado" será "null", en "gol" se apuntará el minuto en el que sucedió, en "tiros recibidos" será el número de tiros que ha recibido el portero apuntado en el valor, en "asistencia" será el id del evento tipo "gol" al que ha asistido en el valor y en "falta" será el minuto en el que ha sucedido y el valor los minutos que ha sido expulsado.
- **TipoEvento.** En esta se guardaran los diferentes tipos de eventos que pueden suceder en un partido.



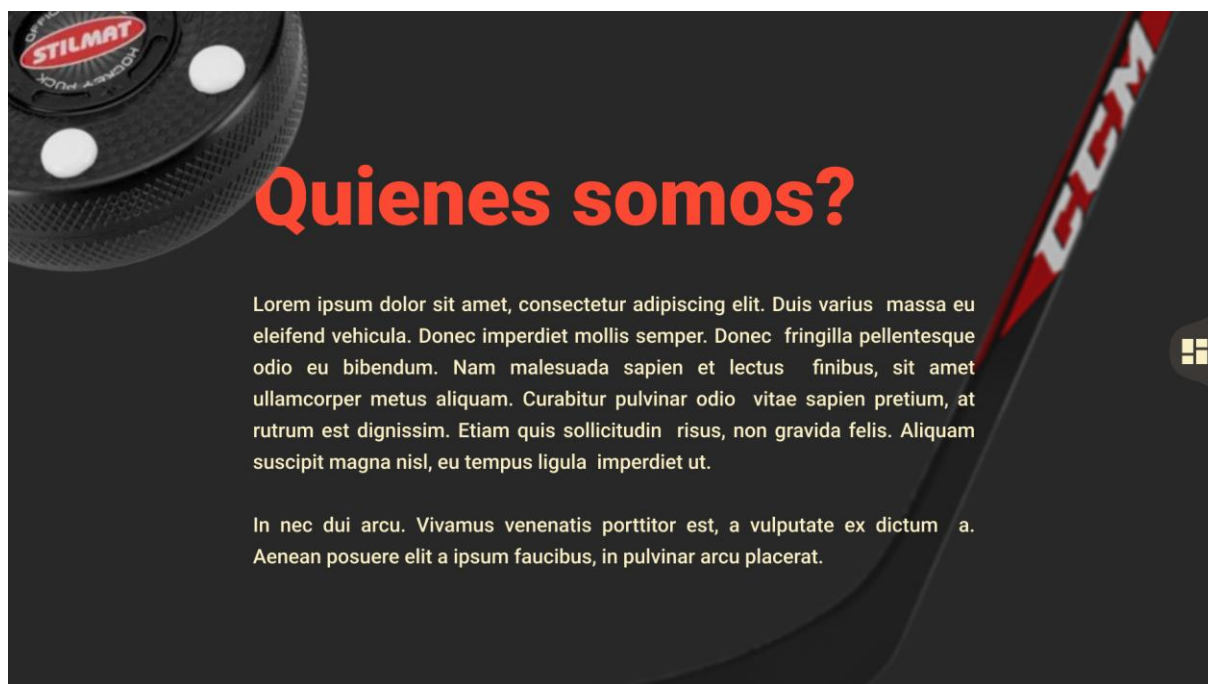
Diseño de la interfaz de usuario

En esta aplicación se ha buscado un diseño sencillo, atractivo y moderno, pensando siempre en su usabilidad y que este se centrará en la temática del hockey línea. Se utilizó, además, una paleta de colores oscura para no dañar la vista de los visitantes y con contraste por la misma razón.

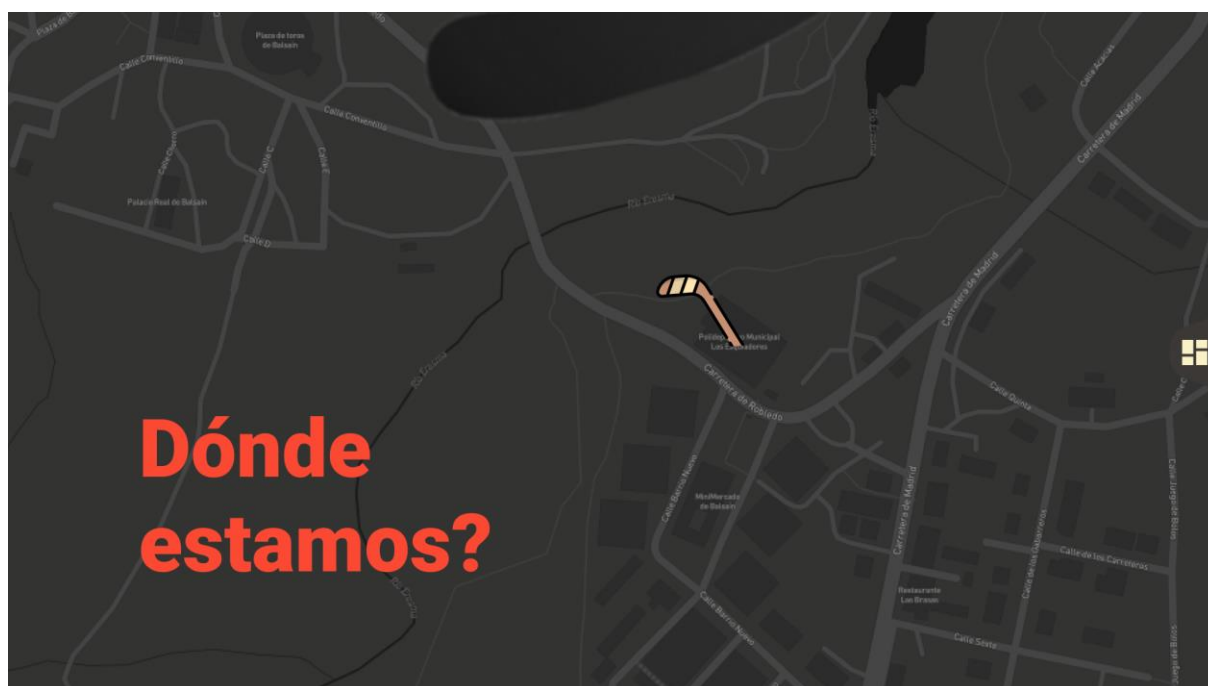
El inicio de la aplicación se buscaba mostrar lo más importante del equipo a primer golpe de vista sin mucha información extra para que el visitante no se vea abrumado, una sencilla identificación en el centro de la pantalla con algo de representación en el fondo mediante una fotografía del equipo:



Después, se mostraría más información en la que utilizamos imágenes de objetos relacionados con el hockey línea para, como se ha mencionado anteriormente, mantener la temática:



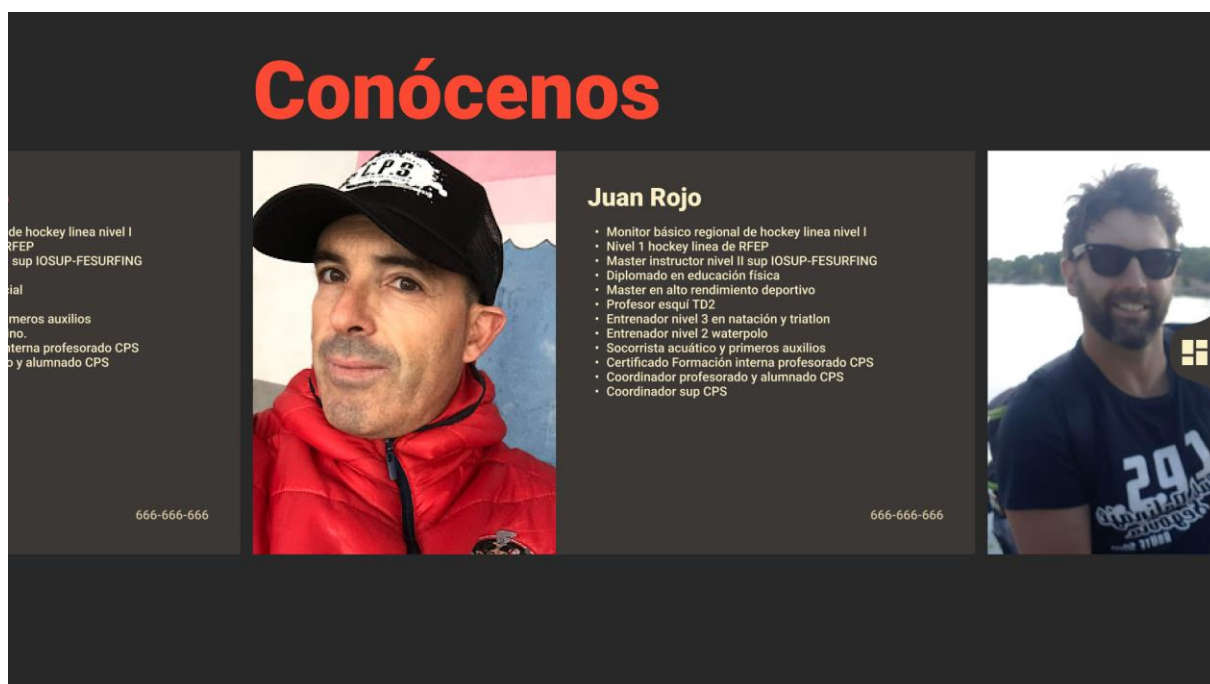
Tras esto, mantenemos la paleta oscura y la sencillez con un simple marcador en la posición que queremos resaltar centrando esta para que sea lo primero a lo que se nos atraiga la mirada tras el título:



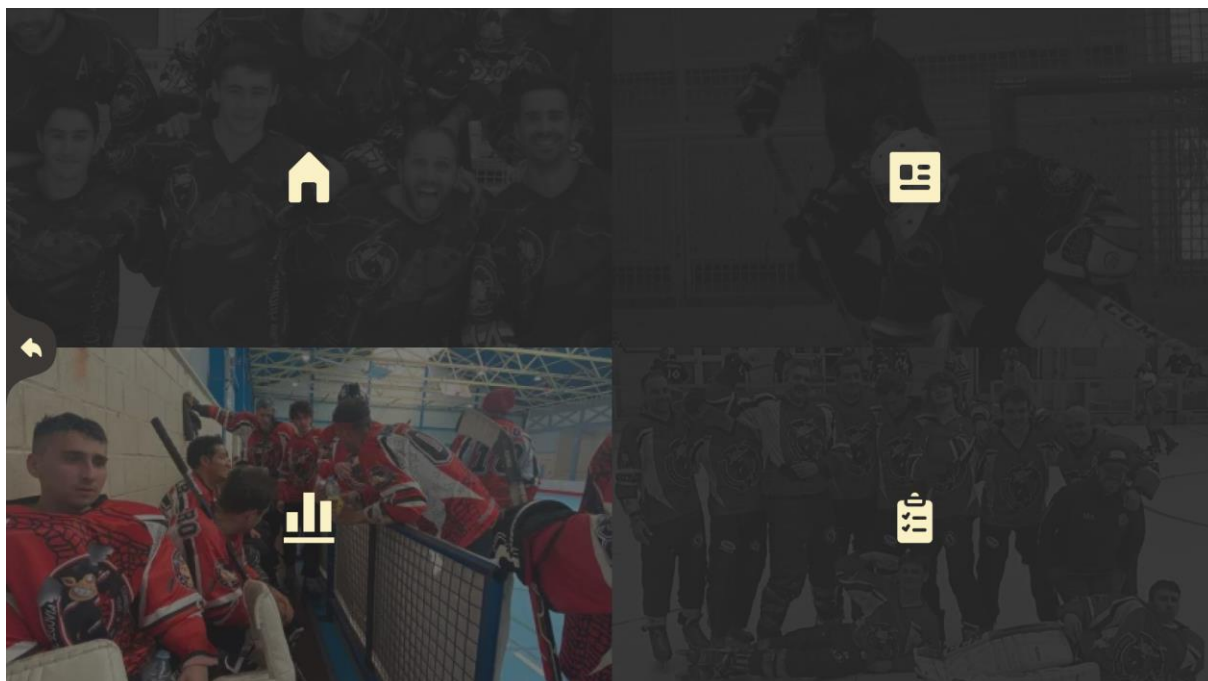
Lo siguiente serían las noticias que mostraremos en forma de periódicos amontonados para darle un toque estético y atractivo:



Y la última parte del inicio en la que nos interesa sencillamente mostrar de manera que se sienta cercana a los trabajadores del equipo, es por esto que las imágenes ocupan la mayoría de la pantalla y el texto en contraste no tenga color y sea de menos tamaño.



En la pantalla de navegación se mostrará las cuatro secciones de las que se compone la aplicación, haciendo que estas ocupen el mayor espacio posible para que sean fáciles de acceder desde cualquier dispositivo; usando además iconos reconocibles de cada apartado.



Dentro de las estadísticas generales o resumen dividimos la pantalla en 4 reconocibles secciones, manteniendo la sencillez y la busca de no sobrecoger al usuario con demasiada información:



COCHINILLOS VOLADORES

Lugar: Segovia

Jugadores: 100

Partidos ganados: 50

Partidos perdidos: 20



Serrano Casado, Jaime

Goles: 0

Asistencias: 0

Puntos: 0

Faltas: 20

M. sanción: 200

	-		Amateur	15/04/2024	10 - 0
	-		Aficionados	15/04/2024	1 - 5
	-		Amateur	15/04/2024	10 - 0
	-		Amateur	15/04/2024	10 - 0
	-		Amateur	15/04/2024	10 - 0

Liga regional alevin

1º

Liga regional amateur

1º

Liga regional aficionados

1º

Diseño de la aplicación

Implementación

Entorno de desarrollo

Html (HyperText Markup Language). Lenguaje estándar utilizado para crear y estructurar páginas web. Utiliza una serie de etiquetas (como <html>, <head>, <body>, <p>, <a>, , etc.) para definir diferentes elementos de la página, como texto, enlaces, imágenes y formularios. Las etiquetas pueden tener atributos que añaden información adicional o estilos a los elementos. HTML es esencial para el diseño web y, junto con CSS y JavaScript, permite crear sitios web interactivos y visualmente atractivos.

CSS (Cascading Style Sheets). Lenguaje utilizado para describir la presentación de un documento HTML. Se utiliza para definir cómo los elementos HTML deben ser mostrados en pantalla, en papel o en otros medios. Se usa para aplicar estilos a los elementos HTML. Define el aspecto y formato de los elementos mediante reglas de estilo que especifican propiedades como color, fuentes, márgenes, relleno, bordes y disposición del contenido. Cada regla CSS tiene un selector que selecciona el elemento HTML a estilizar y un bloque de declaración que contiene una o más declaraciones de estilo. Las hojas de estilo pueden ser internas, dentro de la misma página HTML, externas en un archivo separado, o en línea directamente dentro de una etiqueta HTML.

JavaScript. Lenguaje de programación que se utiliza principalmente para crear contenido dinámico e interactivo en páginas web. Mientras que HTML estructura el contenido y CSS define el estilo, JavaScript permite manipular y controlar ambos para mejorar la experiencia del usuario. Permite a los desarrolladores agregar interactividad a las páginas web, como responder a eventos del usuario (clics, desplazamientos, entradas de teclado), modificar el contenido de la página sin recargarla (a través del DOM), validar formularios, crear animaciones, manejar multimedia y mucho más. Es un lenguaje de programación de alto nivel, interpretado y basado en prototipos.

HTMX. Biblioteca JavaScript que permite a los desarrolladores crear aplicaciones web interactivas y dinámicas utilizando atributos HTML en lugar de escribir JavaScript explícito. Facilita el envío de solicitudes HTTP desde HTML, la actualización de elementos de la página, el manejo de eventos y la interacción con el servidor de manera declarativa. HTMX se enfoca en mejorar la simplicidad y la mantenibilidad del código HTML, permitiendo que gran parte de la lógica de la aplicación se gestione a través de atributos HTML.

```
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo de HTMX</title>
  <script src="https://unpkg.com/htmx.org@1.5.0"></script>
</head>
<body>
  <h1>Ejemplo de HTMX</h1>
  <button hx-get="/mensaje" hx-target="#respuesta">Haz clic aquí</button>
  <div id="respuesta"></div>
</body>
</html>
```

En este ejemplo, el botón tiene un atributo `hx-get` que especifica una solicitud GET al servidor en la URL `/mensaje`. El atributo `hx-target` indica el elemento (`#respuesta`) donde se debe insertar la respuesta del servidor. Cuando el usuario hace clic en el botón, HTMX envía una solicitud GET a `/mensaje` y actualiza el contenido del div con `id="respuesta"` con la respuesta del servidor.

HTMX tiene muchas ventajas como su simplicidad, permitiendo agregar interactividad sin escribir JavaScript adicional, declaratividad, manejando la lógica de la interfaz de usuario a través de atributos HTML, reducción de la complejidad, menos código JavaScript significa menos mantenimiento y menos errores, progresividad, se puede agregar a proyectos existentes sin necesidad de una reescritura completa.

Elegí HTMX por, como se ha comentado antes, su inmensa simplicidad y lo mucho que ayuda a devolver a la web esa idea de que al cliente solamente le serviremos hipertexto, no debemos controlar el estado tanto en cliente como en servidor, nosotros procesamos la información y el cliente sencillamente la muestra.

Rust. Lenguaje de programación de sistemas que se centra en la seguridad, el rendimiento y la concurrencia. Desarrollado por Mozilla, Rust está diseñado para ayudar a los desarrolladores a crear software rápido y seguro, previniendo errores comunes como la gestión incorrecta de la memoria que pueden llevar a fallos y vulnerabilidades de seguridad. Combina las ventajas del control de bajo nivel de lenguajes como C y C++ con características de seguridad que evitan errores comunes en la gestión de memoria. Rust utiliza un sistema de propiedad y préstamos que garantiza la seguridad de memoria y la concurrencia sin necesidad de un recolector de basura, lo que lo hace ideal para aplicaciones de alto rendimiento.

Sus características principales son la seguridad de memoria evitando errores de segmentación y condiciones de carrera mediante su sistema de propiedad y préstamos. Cada valor en Rust tiene un propietario, y las reglas del lenguaje garantizan que no haya referencias colgantes o acceso concurrente no seguro. El rendimiento, ya que compila a código nativo de máquina, lo que permite un rendimiento comparable al de C y C++. La ausencia de un recolector de basura significa que el rendimiento es predecible y no hay pausas de recolección de memoria. La concurrencia, facilita la escritura de código concurrente y seguro. Las herramientas de verificación del compilador aseguran que los datos compartidos entre hilos se manejen de manera segura, reduciendo los errores concurrentes. Y por último, pero en mi opinión su mayor ventaja, el sistema de tipos, tiene un sistema de tipos estático y fuerte que ayuda a detectar errores en tiempo de compilación en lugar de en tiempo de ejecución, aumentando la fiabilidad del código.

Elegí rust ya que es un lenguaje extremadamente rígido, su increíble idea de olvidarse del recolector de basura mediante una filosofía en la que todo debe ser seguro y añadido a un compilador que ayuda al programador a mantener esta en todo momento siendo extremadamente estricto pero al mismo tiempo explícito y claro en el porqué de sus errores hace al lenguaje increíblemente atractivo. Solamente decir que una aplicación tan completa como la que he desarrollado no necesita más de 1GB de RAM y utiliza de manera estable solamente un 2% del procesador con picos máximos de un 3% es algo extremadamente exótico en el mundo de frameworks de JavaScript en el que nos encontramos ahora mismo.

Axum. Framework web basado en Rust que se enfoca en la ergonomía, la modularidad y la seguridad. Está diseñado para aprovechar al máximo las características de seguridad y rendimiento de Rust, proporcionando una forma eficiente y segura de construir aplicaciones web. Es parte del ecosistema Tokio, que es un runtime asíncrono en Rust. Se centra en facilitar la creación de servidores HTTP robustos y seguros, permitiendo a los desarrolladores definir rutas y manejar solicitudes HTTP de manera intuitiva y eficiente. Axum está construido sobre hyper, una biblioteca HTTP de bajo nivel en Rust, y está diseñado para integrarse bien con otras partes del ecosistema Tokio, como tower, que proporciona middleware y abstracciones para servicios.

Sus características principales son su simplicidad y ergonomía utilizando macros y un enfoque basado en funciones para definir rutas y manejar solicitudes, lo que facilita la escritura y comprensión del código, asincronía, aprovechando las capacidades asíncronas de Rust y Tokio, permitiendo manejar grandes cantidades de tráfico de manera eficiente y con bajo consumo de recursos, seguridad y concurrencia, beneficiándose de las garantías de seguridad de memoria y la concurrencia segura de Rust, haciendo que las aplicaciones sean menos propensas a errores como condiciones de carrera y violaciones de memoria, modularidad, se integra fácilmente con otras herramientas y bibliotecas del ecosistema Tokio, como tower, para añadir middleware y otras funcionalidades.

Es por esta integración y gran uso de una de las funcionalidades más potentes del sistema de tipos de rust, los macros, que decidí utilizar el framework Axum. Crear mi propio extractor para poder hacer un sistema de autenticación fue algo sencillo y directo, implementar este middleware en las rutas necesarias para protegerlas fue tan sencillo como escribir una sola línea gracias a su increíble enrutador en el que se pueden añadir las capas que quieras y crear así interceptores en los que hacer lo que necesites con la petición. Implementé un sencillo registro de errores que escribe todos en la consola, pero implementar un registro que se pudiese acceder online sería tarea de unas horas. Y, como he dicho, la implementación hace el acceso a otras herramientas como el gestor asíncrono de archivos de tower algo de lo que no te tienes que preocupar, además de las otras librerías como "axum_htmx" que he llegado a utilizar que están construidas alrededor de este framework y hacen sencillo el trabajar con las otras tecnologías que he utilizado.

Askama. Biblioteca de plantillas para Rust que permite generar contenido HTML (u otros formatos de texto) de manera eficiente y segura. Está inspirada en la sintaxis de Jinja2, una popular biblioteca de plantillas para Python, lo que la hace familiar para aquellos que han trabajado con Jinja2 o similares. Askama está diseñada para ser rápida y segura, aprovechando el sistema de tipos y las características de seguridad de Rust. Permite a los desarrolladores definir plantillas en archivos separados con una sintaxis declarativa y luego renderizarlas con datos proporcionados desde Rust. La biblioteca compila las plantillas en código Rust en tiempo de compilación, lo que mejora el rendimiento y asegura que cualquier error en la plantilla se detecte antes de la ejecución.

Sus características son seguridad de tipos, ya que las plantillas se compilan en código Rust, los errores de tipo y sintaxis se detectan en tiempo de compilación, rendimiento, al ser compiladas se elimina la sobrecarga en tiempo de ejecución, lo que hace que el renderizado de plantillas sea extremadamente rápido, facilidad de uso, la integración con Rust es sencilla y las plantillas se gestionan de manera intuitiva.

He escogido esta tecnología ya que complementa mucho con la idea de servir hipertexto al cliente gracias a la ayuda que ofrece a la hora de construir lo que vamos a servirle. Las plantillas han sido extremadamente potentes a la hora de crear una estructura en mi web, no teniendo que escribir la barra de navegación en cada pantalla, haciendo plantillas para formularios con los que puedo modificar todos los datos de la aplicación, y generando listas de datos a partir de una sencilla lista en rust. Además, esta biblioteca cuenta con una implementación con el framework Axum, con lo que ha sido muy sencillo el utilizar las plantillas con él, ya que al añadir una librería que ayuda con su conexión, “askama_axum”, el servir las plantillas al cliente era tan sencillo como crear un struct y añadirle los campos necesarios y la librería lo convertiría junto a la plantilla en un HTML que se pudiese servir al cliente para que este sencillamente lo muestre.

MySQL. sistema de gestión de bases de datos relacional ampliamente utilizado que permite almacenar, gestionar y recuperar datos de manera eficiente. Utiliza el lenguaje de consulta estructurado (SQL) para realizar operaciones en la base de datos. MySQL es conocido por su rendimiento, confiabilidad y facilidad de uso, y se emplea en una variedad de aplicaciones, desde pequeños proyectos web hasta grandes sistemas empresariales. Permite a los usuarios crear bases de datos, tablas y relaciones entre tablas, y proporciona una amplia gama de operaciones para manipular y consultar los datos almacenados. Es compatible con múltiples sistemas operativos y puede integrarse con diversas tecnologías y lenguajes de programación.

Sus características son escalabilidad y flexibilidad, puede manejar grandes volúmenes de datos y adaptarse a diferentes necesidades de almacenamiento y consulta, alto rendimiento, es capaz de realizar consultas y operaciones de escritura rápidamente, seguridad, tiene robustas características de seguridad, incluyendo autenticación de usuarios y control de acceso, compatibilidad, funciona bien con una amplia variedad de lenguajes de programación y plataformas, lo que lo hace versátil para diferentes tipos de aplicaciones, alta disponibilidad, soporta replicación y clustering para asegurar alta disponibilidad y recuperación ante fallos.

He elegido esta base de datos, además de que es la que estudié en el grado, porque me gusta la idea de las bases de datos relacionales, pienso que son la mejor representación de datos y que es una forma muy ordenada de guardarlos y trabajar con ellos. También podría haber utilizado una base de datos como SQLite pero no es una tecnología con la que esté familiarizado, y ya que la base de datos pienso que es de las piezas más importantes sino la más importante de una aplicación no quería probar nuevas tecnologías para ella en un proyecto tan importante. Además, la herramienta o librería que he utilizado está bastante bien implementada para trabajar con MySQL, pudiendo incluso trabajar con fechas de manera sencilla y teniendo la suficiente documentación para encontrar todo lo que he necesitado durante el desarrollo de la aplicación.

SQLx. Librería asíncrona para interactuar con bases de datos SQL en Rust. A diferencia de otras bibliotecas de bases de datos, SQLx se destaca por ser completamente asíncrona y por proporcionar consultas SQL verificadas en tiempo de compilación, lo que mejora la seguridad y la eficiencia del desarrollo. SQLx es compatible con varias bases de datos, incluidas PostgreSQL, MySQL, SQLite y MSSQL. Permite a los desarrolladores escribir consultas SQL directamente en su código Rust y proporciona verificación en tiempo de compilación de estas consultas. Esto significa que los errores en las consultas SQL se detectan durante la compilación en lugar de en tiempo de ejecución, lo que reduce significativamente la probabilidad de errores en la base de datos.

Sus características son asíncrono, diseñado para trabajar con el ecosistema asíncrono de Rust, utilizando Tokio o async-std como runtime, verificación en tiempo de compilación, asegurándose de que las consultas sean válidas y que los tipos de datos coincidan con los definidos en la base de datos, compatibilidad con múltiples bases de datos como PostgreSQL, MySQL, SQLite y MSSQL, lo que lo hace muy versátil, interfaz intuitiva, esta es fácil de usar a la hora de ejecutar consultas y manejar los resultados.

Esta herramienta ha sido de las que más han ayudado durante el desarrollo, cuenta con todo, puedes descargar su herramienta de terminal “sqlx-cli” con la que puedes crear migraciones, ejecutarlas y controlarlas para mantener todas tus estancias de bases de datos con la misma estructura, contando también con macros para que se prueben al ejecutar el código. Además, sus macros para escribir órdenes MySQL las cuales son comprobadas a tiempo de compilación es una herramienta muy útil que salva mucho tiempo y errores a la hora de estar desarrollando las utilidades de la aplicación. SQLx es un buen ejemplo de una herramienta que implementa una funcionalidad en un lenguaje sin modificar la funcionalidad, ya que puedo seguir utilizando el lenguaje SQL para hacer mis órdenes de la manera que quiero y esto se traducirá al lenguaje que estoy utilizando, en este caso rust, de manera increíble. Como se puede ver en mi código no necesito ni hacer las transformaciones de registros a structs de manera manual sino que la misma herramienta sabe traducirlos mediante los nombres de los campos y además comprobará su tipo de dato y si este puede ser NULL o no lo puede ser, en este caso traduciendo esto al tipo de datos de rust “Option” ya que en rust no existe el término null.

Docker. Plataforma de software que permite a los desarrolladores construir, desplegar y gestionar aplicaciones dentro de contenedores. Los contenedores son unidades ligeras y portátiles que incluyen todo lo necesario para ejecutar una aplicación, incluyendo el código, las dependencias, las bibliotecas y el sistema operativo. Esto garantiza que la aplicación se ejecute de manera consistente en cualquier entorno, ya sea en la máquina local del desarrollador, en un servidor de prueba o en producción. Los contenedores son similares a las máquinas virtuales, pero son más ligeros y eficientes porque comparten el núcleo del sistema operativo en lugar de tener un sistema operativo completo para cada instancia. Esto permite un uso más eficiente de los recursos y una mayor rapidez en el arranque de las aplicaciones.

Sus características son portabilidad, los contenedores Docker pueden ejecutarse en cualquier sistema que soporte Docker, lo que garantiza que la aplicación funcione de manera consistente en diferentes entornos, eficiencia, los contenedores comparten el núcleo del sistema operativo y otros recursos, lo que los hace más ligeros y rápidos que las máquinas virtuales, aislamiento, cada contenedor ejecuta una aplicación de manera aislada, lo que mejora la seguridad y la gestión de dependencias, escalabilidad, facilita la creación y gestión de entornos de desarrollo, prueba y producción, y permite escalar aplicaciones de manera eficiente.

En este proyecto se ha utilizado para poder aprovechar la idea de contener una aplicación en un entorno que sabemos que funciona, no importa donde se ejecute, mientras pueda ejecutar docker va a poder ejecutar nuestro código. Esto sumado al plugin que he utilizado, “compose”, el cual ayuda con el trabajar con más de un contenedor, ha facilitado el desplegar y reproducir mi código en cualquier sitio. No solamente eso sino que también ha ayudado con simular diferentes partes necesarias como una base de datos para probar mientras desarrollaba sin tener la necesidad de que estos datos se escribieran de manera permanente en mi ordenador corriendo un servidor de MySQL en él. Gracias a esta herramienta he escrito una sola vez mi entorno y no importa los cambios que haga se que va a correr en el servidor de manera correcta, además de que las herramientas que he utilizado de CI/CD se implementan de manera increíble con Docker. Y algo también increíblemente útil de Docker son las incontables imágenes que puedes encontrar en su comunidad para simular cualquier entorno para una herramienta en concreto.

En el Dockerfile de la aplicación lo que estamos haciendo es primero cargar una imagen oficial de rust con la versión de debian bullseye en la que configuraremos la variable de entorno “SQLX_OFFLINE”, instalamos la herramienta para compilar nuestro código llamada “cargo-chef” y declaramos la carpeta en la que trabajaremos “/cochinillos-voladores”

```
FROM rust:bullseye as chef
ENV SQLX_OFFLINE=true
RUN cargo install cargo-chef
WORKDIR /cochinillos-voladores
```

Después prepararemos un JSON para que la compilación de nuestro código sea aún más rápida.

```
FROM chef AS planner
COPY . .
RUN cargo chef prepare --recipe-path recipe.json
```

Tras esto, utilizaremos lo que hemos preparado anteriormente para acelerar la compilación de nuestro código y finalmente lo compilamos en un binario llamado “cochinillos-voladores”.

```
FROM chef AS builder
COPY --from=planner /cochinillos-voladores/recipe.json recipe.json
RUN cargo chef cook --release --recipe-path recipe.json
COPY . .
RUN cargo build --release --bin cochinillos-voladores
```

Por último, cargaremos una nueva imagen oficial de rust con debian bullseye y copiaremos tanto el binario que hemos compilado como los assets estáticos que necesita el binario para cargar, pudiendo ser imágenes, css, etc. Para que esto sea accesible abriremos el puerto 3000 y haremos visible el binario que hemos compilado.

```
FROM rust:bullseye AS runtime
WORKDIR /cochinillos-voladores
COPY --from=builder /cochinillos-voladores/target/release/cochinillos-voladores /cochinillos-voladores
COPY --from=builder /cochinillos-voladores/assets /cochinillos-voladores/assets
ENTRYPOINT ["/cochinillos-voladores/cochinillos-voladores"]
EXPOSE 3000
```

Para correr este archivo junto a una base de datos utilizaremos el plugin “compose” de docker, para esto realizaremos dos nuevos archivos, uno para el código y la base de datos y otro para el servidor de redirección “Caddy”. En este primero lo que haremos será dividir el archivo en servicios, el primero siendo la api en la que utilizaremos la imagen que subimos en el workflow de github a docker hub y configuraremos algunas cosas como las variables de entorno, el contexto, el puerto, sus dependencias, un volumen para mantener las imágenes que suben los usuarios y no se pierdan con cada vez que se apague y encienda el contenedor.

```
services:
  api:
    container_name: api
    image: alejandros2i/cochinillos-voladores
    environment:
      DATABASE_URL: "mysql://root:${MYSQL_PASSWORD}@db:3306/cochinillos"
      RUTA_UPLOADS: ${RUTA_UPLOADS}
      MAIL: ${MAIL}
      USUARIO_MAIL: ${USUARIO_MAIL}
      CONTRA_MAIL: ${CONTRA_MAIL}
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    depends_on:
      - db
    volumes:
      - uploads:/cochinillos-voladores/uploads
    networks:
      - app_network
```

Después configuraremos el contenedor para la base de datos en el que utilizaremos la imagen oficial de mysql para el contenedor y configuraremos algunas variables de entorno, el puerto y el volumen donde se mantendrán todos los datos sin que se borren como en el anterior contenedor. Además de la red que hemos configurado en el anterior contenedor y que utilizaremos para conectar todos los contenedores entre sí.

```
db:
  container_name: db
  image: mysql:latest
  restart: always
  environment:
    MYSQL_DATABASE: cochinillos
    MYSQL_ROOT_PASSWORD: ${MYSQL_PASSWORD}
  ports:
    - "3306:3306"
  volumes:
    - db:/var/lib/mysql
  networks:
    - app_network
```

Por último configuraremos los volúmenes que hemos declarado en los dos anteriores contenedores para que se asegure de su existencia, en el que podríamos configurar algunas opciones pero las iniciales son las que necesitamos, y además configuraremos la red que utilizaremos para conectar con todos los contenedores.

```
volumes:
  db:
  uploads:

networks:
  app_network:
```

Por último, en otro archivo a parte configuré el servidor de redirección en el que utilizaremos la imagen oficial que nos ofrece Caddy, que esta vez corre en un debian alpine, en la que configuraremos que no se reinicie a no ser que el contenedor sea parado en algún momento, los puertos 443 y 80 ya que estos son los relacionados con http y https, el acceso al archivo de configuración del servidor caddy “Caddyfile” y dos volúmenes en los que guardará sus datos permanentes. Conectaremos también este contenedor a la red que hemos conectado los dos anteriores contenedores para que así los tres puedan comunicarse y verse entre sí.

```
services:
  caddy:
    container_name: caddy
    image: caddy/caddy:2.8-alpine
    restart: unless-stopped
    ports:
      - 443:443
      - 80:80
    volumes:
      - $PWD/Caddyfile:/etc/caddy/Caddyfile
      - caddy_data:/data
      - caddy_config:/config
    networks:
      - app_network

volumes:
  caddy_data:
  caddy_config:
```

GitHub. Plataforma de desarrollo colaborativo y hospedaje de código que utiliza el sistema de control de versiones Git. Ofrece a los desarrolladores una variedad de herramientas para trabajar juntos en proyectos de software, gestionar versiones del código, revisar cambios, colaborar en código fuente y documentar sus proyectos. GitHub es ampliamente utilizado tanto por proyectos de código abierto como por proyectos privados de empresas y organizaciones. Permite a los desarrolladores almacenar sus repositorios de código en la nube, facilitando el acceso y la colaboración desde cualquier lugar. Los desarrolladores pueden clonar repositorios, hacer cambios en el código, crear ramas (branches) para desarrollar nuevas características de manera aislada, y luego fusionar (merge) esos cambios en la rama principal. GitHub también proporciona herramientas para la gestión de proyectos, seguimiento de problemas (issues), integración continua (CI), y despliegue continuo (CD).

Sus características son control de versiones con git, lo que permite a los desarrolladores rastrear cambios, revertir a versiones anteriores y trabajar en diferentes ramas simultáneamente, colaboración, se puede colaborar fácilmente en proyectos, utilizando herramientas como pull requests y revisiones de código para discutir y revisar cambios, hospedaje de repositorios, ofrece almacenamiento gratuito para proyectos de código abierto y opciones de pago para proyectos privados, gestión de proyectos, incluye herramientas para gestionar tareas, seguimiento de problemas, y planificación de proyectos, integración con herramientas CI/CD, se integra con muchas herramientas de integración y despliegue continuo, facilitando la automatización de pruebas y despliegues, documentación y wikis, los repositorios pueden incluir archivos README, wikis y documentación detallada para ayudar a los desarrolladores a entender y contribuir a los proyectos.

He utilizado GitHub ya que esta herramienta es utilizada entre todos los desarrolladores y empresas de todo el mundo, es una herramienta muy útil para mantener tu código centralizado y guardado de manera sencilla y rápida sin tener ninguna preocupación de perderlo, además de poder controlar sus cambios y así poder volver en caso de hacer algo no deseado. Además, en este proyecto se ha utilizado su herramienta GitHub actions, la cual se ha explicado en el despliegue, que ha ayudado a la hora de desarrollar el CI/CD inmensamente.

Caddy. Servidor web multiplataforma conocido por su simplicidad, facilidad de configuración y capacidades automatizadas de administración de certificados SSL/TLS. Es especialmente popular por su capacidad de gestionar automáticamente la adquisición y renovación de certificados SSL a través de “Let's Encrypt”, lo que facilita la implementación de sitios web seguros sin necesidad de configuraciones complejas. Destaca por su sencilla configuración, utilizando un archivo de configuración llamado Caddyfile. Este archivo permite definir fácilmente sitios, proxies reversos y otras configuraciones de servidor web con una sintaxis clara y concisa. Caddy es ideal para desarrolladores que buscan una solución de servidor web rápida de configurar y mantener, con características avanzadas integradas de manera predeterminada.

Elegí Caddy por su gran sencillez cuando lo único por lo que lo necesitaba era redirigir las peticiones a un puerto en específico y asegurarme que esto se hiciese de manera que soportase https. Con Caddy es tan sencillo que este archivo es toda la configuración que hice:

```
cochinillosvoladores.org {  
    reverse_proxy api:3000 {  
        header_down Strict-Transport-Security max-age=31536000;  
    }  
}  
  
www.cochinillosvoladores.org {  
    reverse_proxy api:3000 {  
        header_down Strict-Transport-Security max-age=31536000;  
    }  
}
```

Este archivo redirige las peticiones provenientes de “cochinillosvoladores.org” y “www.cochinillosvoladores.org” a un proxy inverso apuntando a el código que estamos corriendo en un contenedor, además añade la cabecera de “Strict-Transport-Security” y una vida máxima de 31.536.000 segundos, o lo que es lo mismo, un año.

Estructura del código

En esta aplicación se ha intentado seguir un diseño parecido al MVC. Se creó un modelo en el que manejamos todo lo relacionado con la creación, modificación y eliminación de los datos en la base de datos, esto incluye todos los errores que pueden surgir en estos procesos y otros controles necesarios. Luego se desarrollaron rutas que se dividieron en dos, rutas de web, en las que se sirven sencillamente páginas web o relacionados, y rutas de api, las cuales se utilizan para comunicarse con la capa de modelo para crear, modificar u eliminar datos. Además, todo el control relacionado con modificar peticiones, acceder a ellas o obtener información necesaria se ha realizado a partir de middleware, en el que todas las peticiones si lo necesitan pasarán por un código que les ofrece la información o las controla de manera sencilla y unificada por toda la aplicación.

Esta última parte es algo que encontré bastante ingenioso y útil, gracias al increíble sistema de tipos de rust y su implementación en el framework Axum conseguí hacer un extractor de manera sencilla para tener siempre que lo necesitase acceso al contexto de la aplicación, en este caso al usuario que está haciendo la petición. Esto lo podemos observar en cualquier controlador de una ruta en la aplicación, como por ejemplo en el de perfil cuando sí existe contexto, el ctx es Some, significa que el usuario tiene la sesión iniciada y que podemos redirigirlo a su perfil con sus datos, en cambio si no tiene contexto, ctx es None, significa que el usuario no tiene una sesión iniciada y le redirigiremos a la pantalla de inicio de sesión.

```
async fn perfil(
    State(_cm): State<ControladorModelo>,
    ctx: Option<Ctx>,
) -> impl IntoResponse {
    match ctx {
        Some(ctx) => PerfilTemplate { usuario: ctx.usuario() }.into_response(),
        None => (HxRedirect(Uri::from_static("/login")), Redirect::to("/login")).into_response()
    }
}
```

Esto lo conseguimos primero con un middleware en el que obtenemos las cookies de la petición, las cuales se mandan siempre que se hace una petición al servidor si existen, y obtenemos el contexto a partir de estas, resolviendo las cookies con otra función y luego añadiendo este contexto a las extensiones de la petición donde podremos accederlas desde otro punto de la aplicación.

```
pub async fn mw_resolver_ctx(
    State(cm): State<ControladorModelo>,
    cookies: Cookies,
    mut req: Request<Body>,
    next: Next,
) -> Response {
    let result_ctx = resolver_ctx(cm, &cookies).await;

    if result_ctx.is_err()
        && !matches!(result_ctx, Err(CtxExtError::NoTokenEnCookies))
    {
        cookies.remove(Cookie::named(AUTH_TOKEN));
    }

    req.extensions_mut().insert(result_ctx);

    next.run(req).await
}
```

En el resolvedor de contexto lo que haremos será conseguir el UUID de la cookie que nos ha pasado la petición mediante un parseo de esta, con este UUID conseguiremos el login de la base de datos que lo contenga y comprobaremos su fecha de caducidad por si ha caducado poder borrar el login y la cookie. En caso contrario si toda validación es pasada entonces lo que haremos es conseguir el usuario relacionado con el login que acabamos de conseguir y lo devolveremos.


```

async fn resolver_ctx(cm: ControladorModelo, cookies: &Cookies) -> CtxExtResult {
    let uuid = Uuid::parse_str(cookies
        .get(AUTH_TOKEN)
        .map(|c| c.value().to_string())
        .ok_or(CtxExtError::NoTokenEnCookies)?.as_str())
        .map_err(|_| CtxExtError::TokenFormatoIncorrecto)?;

    let login = ControladorLogin::login_uuid(cm.clone(), uuid).await
        .map_err(|err| CtxExtError::ErrorModelo(err.to_string()))?
        .ok_or(CtxExtError::LoginNoEncontrado)?;

    // Validar token
    if login.fechaCaducidad <= OffsetDateTime::now_utc().date() {
        ControladorLogin::eliminar_login(cm.clone(), uuid).await
            .map_err(|err| CtxExtError::ErrorModelo(err.to_string()))?;

        let mut cookie = Cookie::from(AUTH_TOKEN);
        cookie.set_path("/");
        cookies.remove(cookie);

        return Err(CtxExtError::TokenExpirado);
    }

    let usuario = ControladorUsuario::usuario_id(cm, login.idUsuario).await
        .map_err(|err| CtxExtError::ErrorModelo(err.to_string()))?
        .ok_or(CtxExtError::UsuarioNoEncontrado)?;

    Ok(Ctx::new(usuario))
}

```

Por último, lo que hice es implementar un nuevo trait al struct que creé en otro lado de la aplicación para poder añadir la funcionalidad de “FromRequestParts” que necesita para que Axum lo ejecute como un extractor en una función.

```

#[async_trait]
impl<S: Send + Sync> FromRequestParts<S> for Ctx {
    type Rejection = Error;

    async fn from_request_parts(parts: &mut Parts, _state: &S) -> Result<Self> {
        parts
            .extensions
            .get:::<CtxExtResult>()
            .ok_or(Error::CtxExt(CtxExtError::NoCtxEnExtension))?
            .clone()
            .map_err(Error::CtxExt)
    }
}

```

Pruebas

Para las pruebas de esta aplicación se ha utilizado pruebas de unidad las cuales se ejecutan durante el workflow de CI/CD, estas son sencillas de implementar ya que las herramientas que se han utilizado ofrecen grandes funcionalidades para hacer del probar la aplicación algo sencillo. Este es el ejemplo de una simple prueba que se puede encontrar en el código de la aplicación en el que podemos observar la primera funcionalidad que hemos utilizado y es de SQLx ya que esta librería nos presenta la oportunidad de ejecutar una base de datos temporal para nuestras pruebas y ejecutar órdenes SQL aparte llamadas “fixtures” donde podemos meter datos de prueba. Entonces podremos comprobar si la función de conseguir un usuario por su id funciona correctamente mediante una sencilla llamada a esta y ver si nos devuelve el resultado esperado.

```
#[sqlx::test(fixtures("usuarios"))]
async fn obtener_usuario(pool: MySQLPool) -> Result<()> {
    let cm = ControladorModelo::new(pool).await?;

    let usuario = ControladorUsuario::usuario_id(cm, 1).await
        .unwrap_or_else(|err| panic!("No se ha conseguido el usuario 1"))
        .ok_or_else(|| panic!("No se ha conseguido el usuario 1")).unwrap();

    assert_eq!(usuario.id, 1);

    Ok(())
}
```

Esto suele desarrollarse para todas las funcionalidades de la aplicación, incluso Axum nos ofrece una herramienta para trabajar con peticiones de prueba de manera muy sencilla. Este es un ejemplo en el que hacemos una simple petición a la pantalla de inicio.

```
let response = app
    .oneshot(Request::builder().uri("/").body(Body::empty()).unwrap())
    .await
    .unwrap();
```

Manual de usuario

Cuando el usuario acceda a la aplicación web la primera pantalla que observará será la portada en la que le mostraremos tanto el nombre del equipo como su logo y una imagen de este de fondo.



Al bajar podrá observar un párrafo en el que se le informa sobre el equipo en detalle.

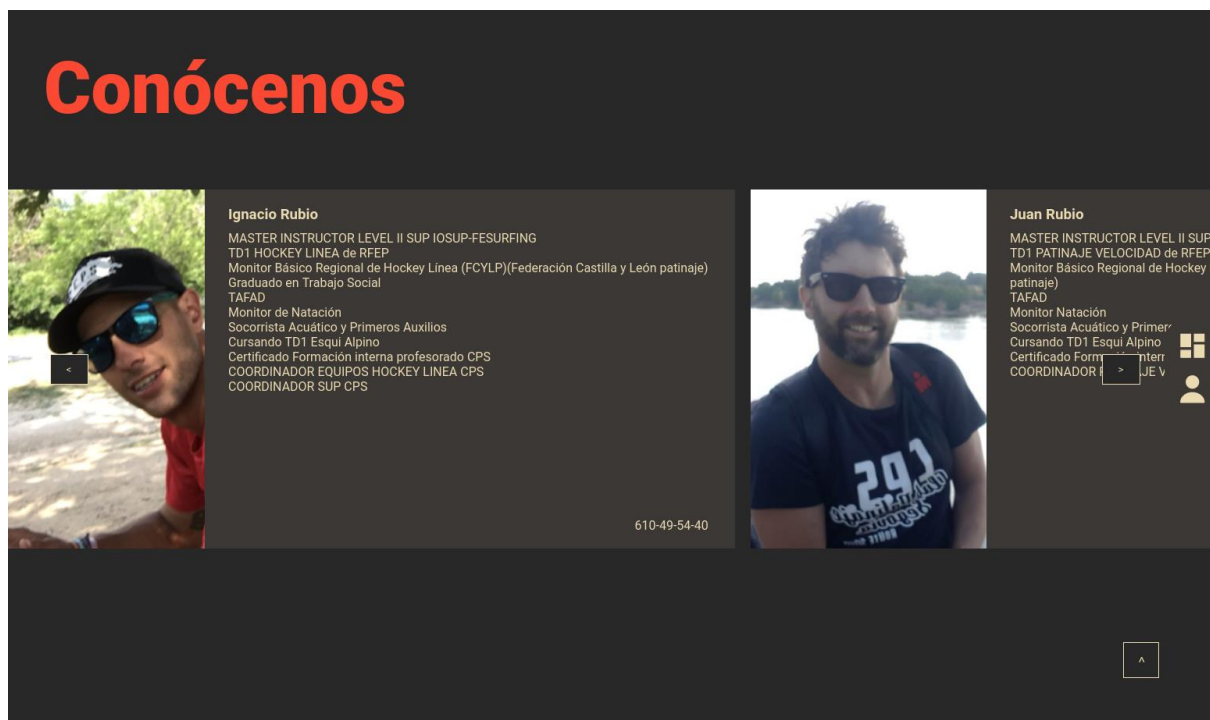


Dónde estamos?

The map shows a network of streets including Calle Conventillo, Calle B, Calle D, Calle F, Calle G, Calle H, Calle I, Calle J, Calle K, Calle L, Calle M, Calle N, Calle O, Calle P, Calle Q, Calle R, Calle S, Calle T, Calle U, Calle V, Calle W, Calle X, Calle Y, and Calle Z. A red location pin is placed on Calle F, near the intersection with Calle G. The text 'Dónde estamos?' is written in large, bold, red letters across the top of the map.

A digital newspaper layout for 'Noticias' is displayed on a dark grey background. The newspaper has a light beige background with a large, bold, black title 'Noticias' centered at the top, flanked by horizontal lines. Below the title, the date '2024-05-05' is printed in a small, black font. The main content area features a photograph of a hockey game in progress, showing a player in a dark jersey and helmet attempting to score a goal, while a goalie in a white and red uniform is blocking the shot. The photo is set within a rectangular frame. To the right of the photo, the article title 'Cuarta jornada de amateur' is written in a bold, black font, followed by a paragraph of text: 'Estamos orgullosos de nuestros jugadores de amateur que han ganado uno de los partidos de la cuarta jornada de la liga con una increíble actuación y una especial enhorabuena a nuestro portero Miguel'. The layout includes several navigation elements: a small grid icon and a person icon on the right side, and a set of three square buttons at the bottom center containing the symbols '<', '>', and 'À'.

Por último, en la sección más abajo se encuentra la información y el contacto de los trabajadores del equipo por si el visitante quiere informarse o incluso contactarlos.



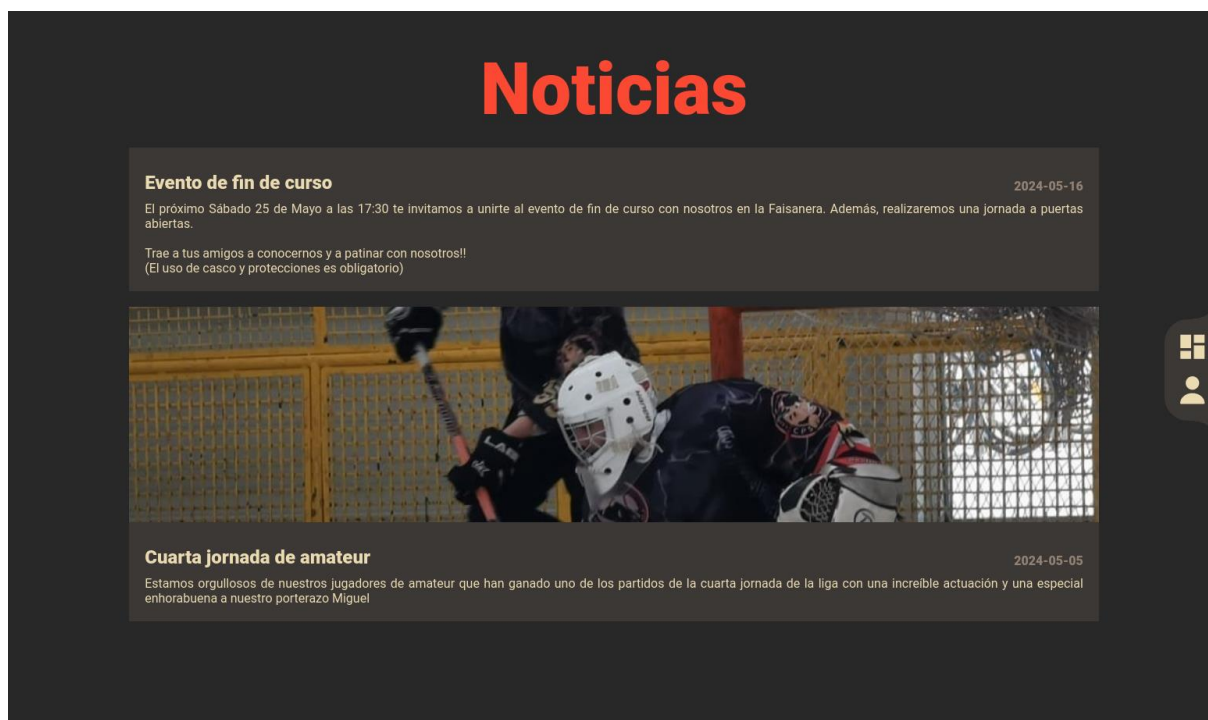
Se podrá acceder al panel en el que podremos ver todas las páginas de la aplicación mediante el botón situado en la barra de navegación en la parte derecha de la pantalla con cuatro cuadrados.



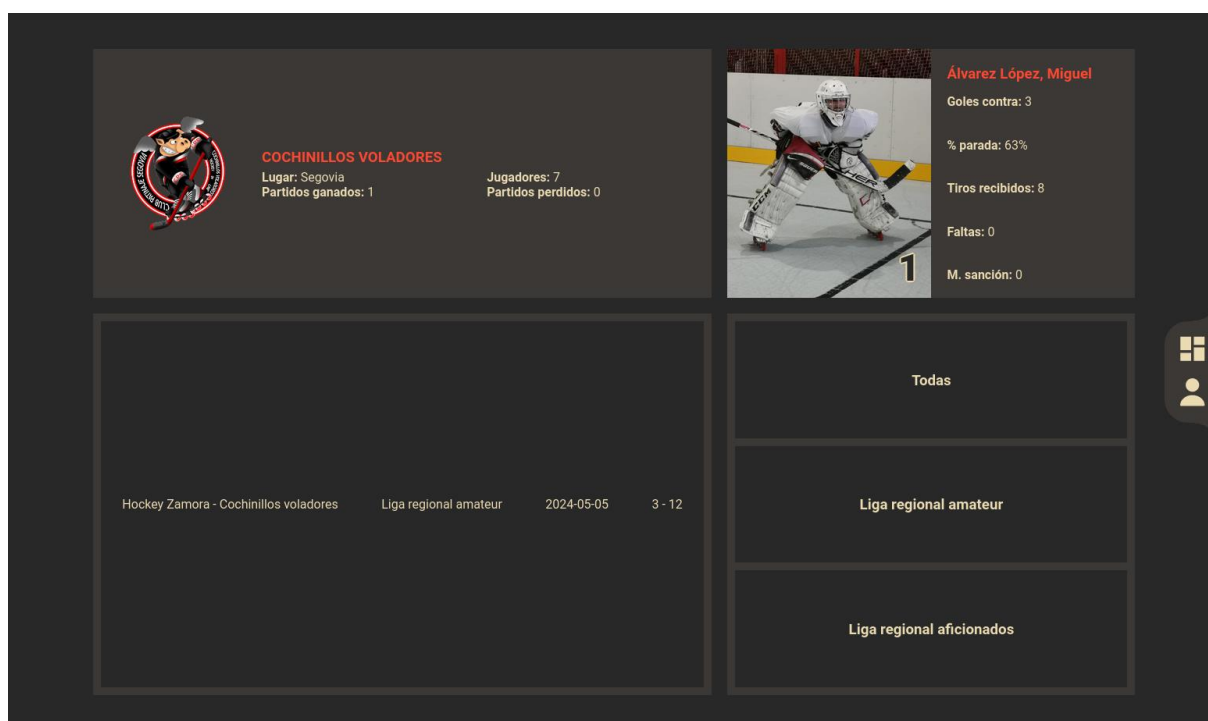
Esto presentará al usuario con las cuatro páginas de la web, siendo estas respectivamente inicio, noticias, estadísticas y formulario de inscripción.



En la página de noticias podemos encontrar un listado de todas las noticias del equipo ordenadas por orden cronológico.



En el apartado de estadísticas podemos encontrarnos en un principio un resumen de las estadísticas del equipo, mostrando en la parte superior izquierda los datos del equipo de forma general, en la parte superior derecha los datos de un jugador aleatorio cada vez que la página es cargada, en la parte inferior izquierda un listado con todos los partidos ordenados cronológicamente y en la parte inferior derecha un listado de todas las competiciones por las que podemos filtrar el listado de partidos. El listado de competiciones filtrará los partidos al hacer clic a la competición deseada, mostrando la opción de “Todas” por defecto donde se mostrarán los partidos de todas las competiciones. En el caso de querer observar las estadísticas específicas de un partido se hará clic en este para que se le redirija a las respectivas estadísticas.



Dentro de las estadísticas de un partido podemos observar en la parte superior izquierda los datos generales de este partido, en la derecha observaremos el listado de todos los eventos ordenados por el minuto en el que sucedieron y en la parte inferior izquierda observaremos el listado de jugadores que tomaron parte en este partido separados por equipo y ordenados por número, mostrando las estadísticas de cada uno en este partido.

Competición: Liga regional amateur

Lugar: Polideportivo barrios

Fecha: 2024-05-05

Hockey Zamora

3 - 12

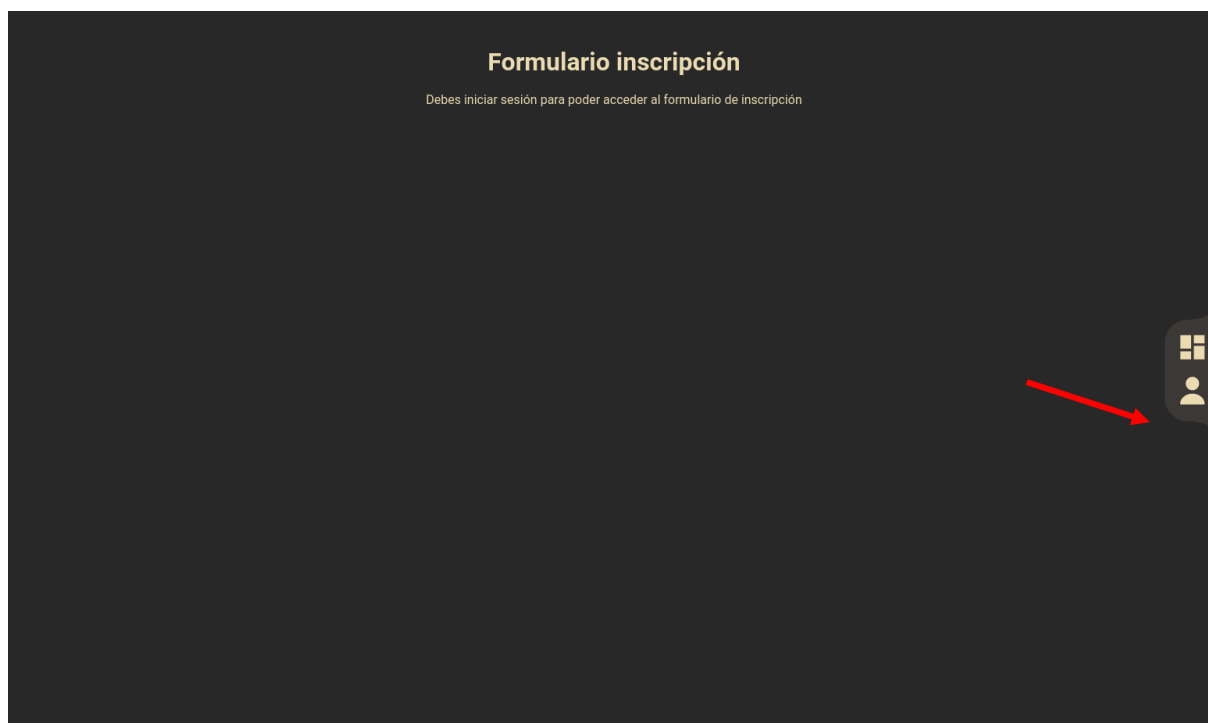
Cochinillos voladores

Hockey Zamora

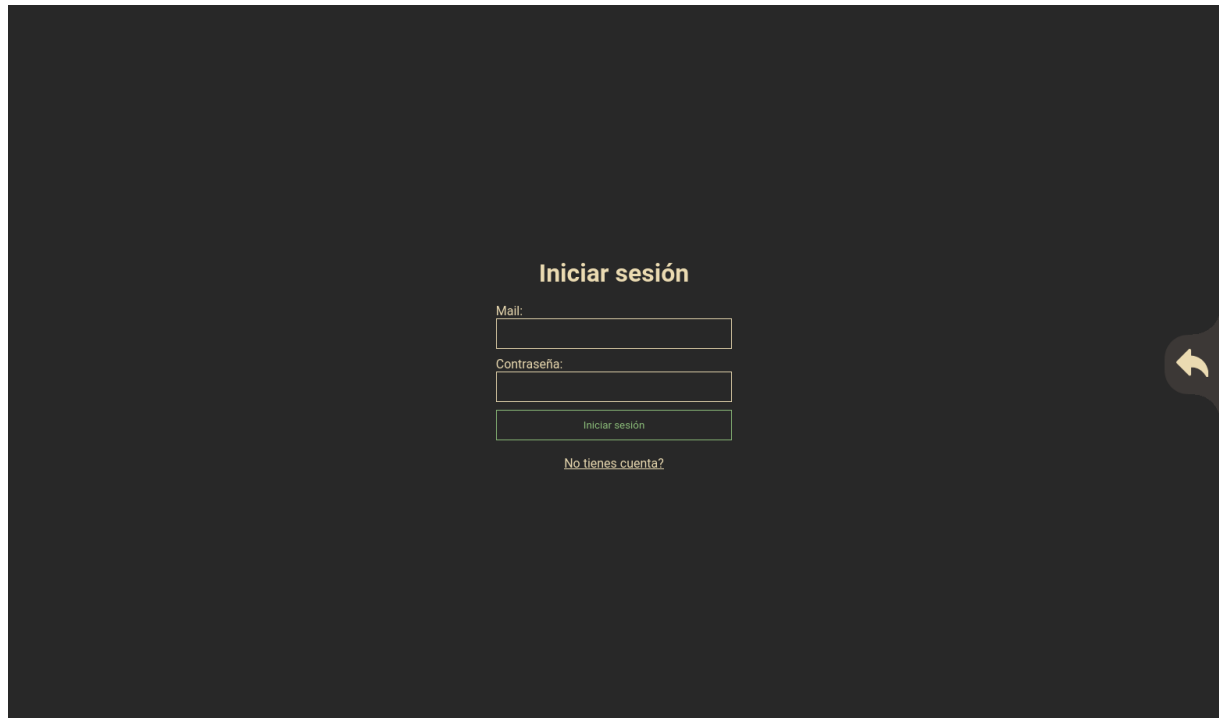
1 Olea Gutiérrez, David	Goles en contra: 12	Tiros recibidos: 34	% parada: 65%	
2 Aparicio Hernández, Javier	Goles: 0	Asistencias: 0	Puntos: 0	Faltas: 1
10 Calvo Cuesta, Izan	Goles: 1	Asistencias: 0	Puntos: 1	Faltas: 0
19 Miñambres Carnero, Tristan	Goles: 1	Asistencias: 0	Puntos: 1	Faltas: 0
23 Prieto Sánchez, Hugo	Goles: 1	Asistencias: 0	Puntos: 1	Faltas: 0
Cochinillos voladores				
1 Álvarez López, Miguel	Goles en contra: 3	Tiros recibidos: 8	% parada: 63%	
7 Rubio Gómez, Ignacio	Goles: 6	Asistencias: 0	Puntos: 6	Faltas: 0
22 Torres Gonzalez, Raúl	Goles: 1	Asistencias: 0	Puntos: 1	Faltas: 0
48 Martín Carreras, David	Goles: 0	Asistencias: 0	Puntos: 0	Faltas: 0
55 Serrano Casado, Jaime	Goles: 2	Asistencias: 0	Puntos: 2	Faltas: 0

0:00:13.0	Gol
Serrano Casado, Jaime	
0:01:13.0	Gol
Torres Gonzalez, Raúl	
0:01:57.0	Gol
Rubio Gómez, Ignacio	
0:03:55.0	Gol
Rubio Gómez, Ignacio	
0:08:48.0	Gol
Calvo Cuesta, Izan	
0:11:01.0	Gol
Miñambres Carnero, Tristan	
0:12:06.0	Gol
Rubio Gómez, Ignacio	
0:19:34.0	Falta
Aparicio Hernández, Javier	
0:20:39.0	Gol
Serrano Casado, Jaime	
0:20:41.0	Gol
Rubio Gómez, Ignacio	
0:21:04.0	Gol

En la página de formulario de inscripción si no estamos iniciados en ninguna sesión observaremos que no podremos realizarlo. Podemos iniciar sesión desde cualquier punto de la aplicación mediante el botón a mano derecha de una persona, accediendo así al inicio de sesión o perfil dependiendo de si estamos o no iniciados.



Al acceder al inicio de sesión se nos mostrará esta pantalla en la que accederemos a nuestra cuenta si contamos con ella mediante el mail y la contraseña. En caso de no contar con una cuenta haremos clic en “No tienes cuenta?” para registrarnos.

A screenshot of a login form on a dark background. The form is centered and contains the title "Iniciar sesión" in bold. Below the title are three input fields: "Mail:", "Contraseña:", and a button labeled "Iniciar sesión". At the bottom of the form is a link that says "No tienes cuenta?". To the right of the form, there is a yellow arrow pointing left, indicating a back button.

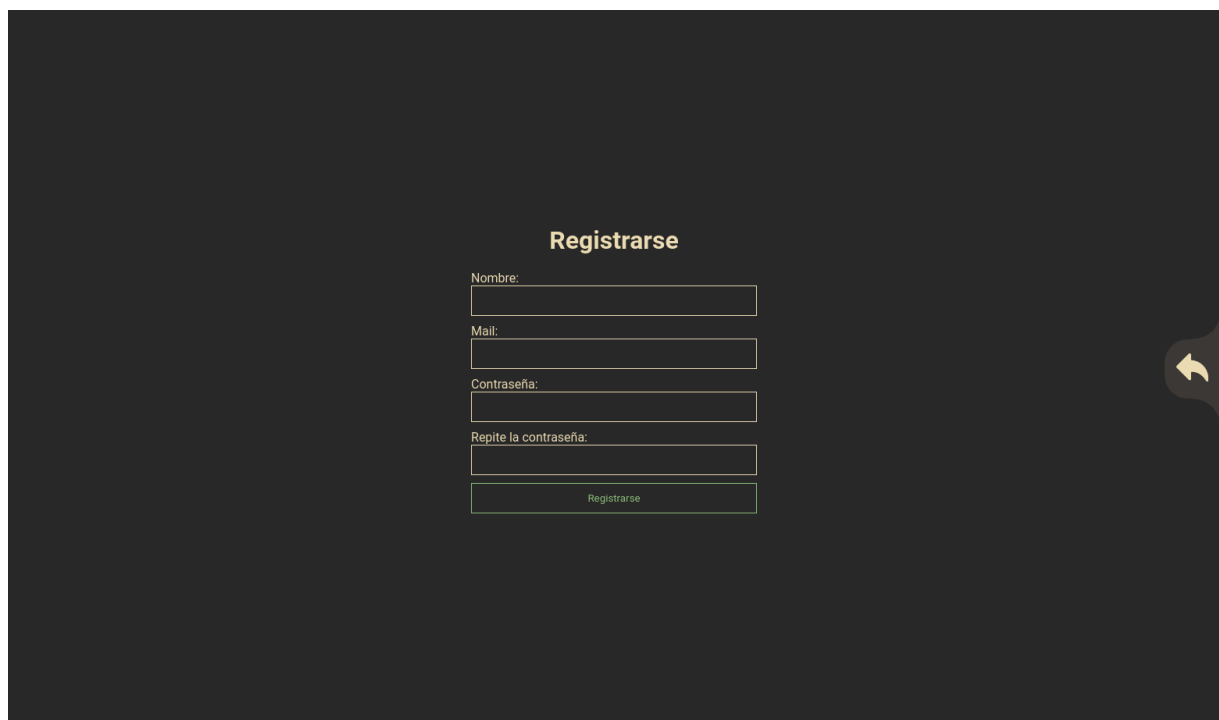
Iniciar sesión

Mail:

Contraseña:

[No tienes cuenta?](#)

Al acceder a registrarnos se nos presentará esta pantalla en la que deberemos rellenar los datos suficientes para crear una nueva cuenta.

A screenshot of a registration form on a dark background. The form is centered and contains the title "Registrarse" in bold. Below the title are five input fields: "Nombre:", "Mail:", "Contraseña:", "Repite la contraseña:", and a button labeled "Registrarse". To the right of the form, there is a yellow arrow pointing left, indicating a back button.

Registrarse

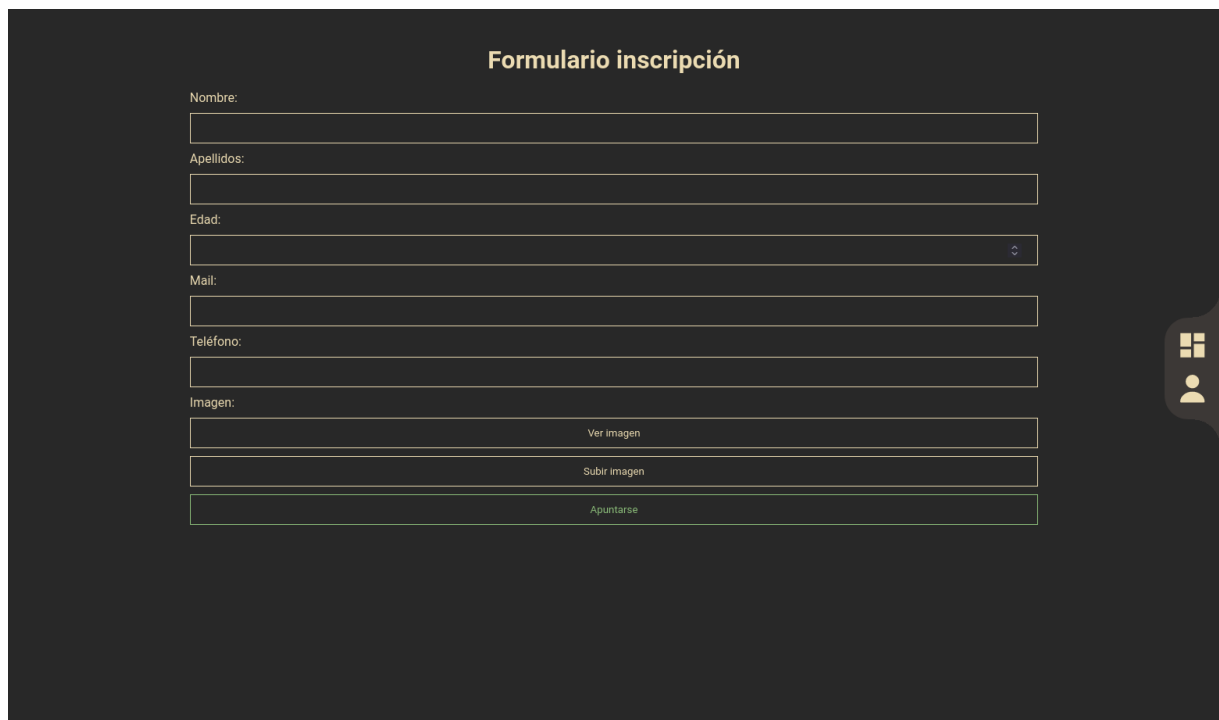
Nombre:

Mail:

Contraseña:

Repite la contraseña:

Cuando iniciemos sesión o nos registremos podremos volver a la pantalla de formulario y esta ya nos ofrecerá la opción de inscribirnos rellenando los campos necesarios.



Formulario inscripción

Nombre:

Apellidos:

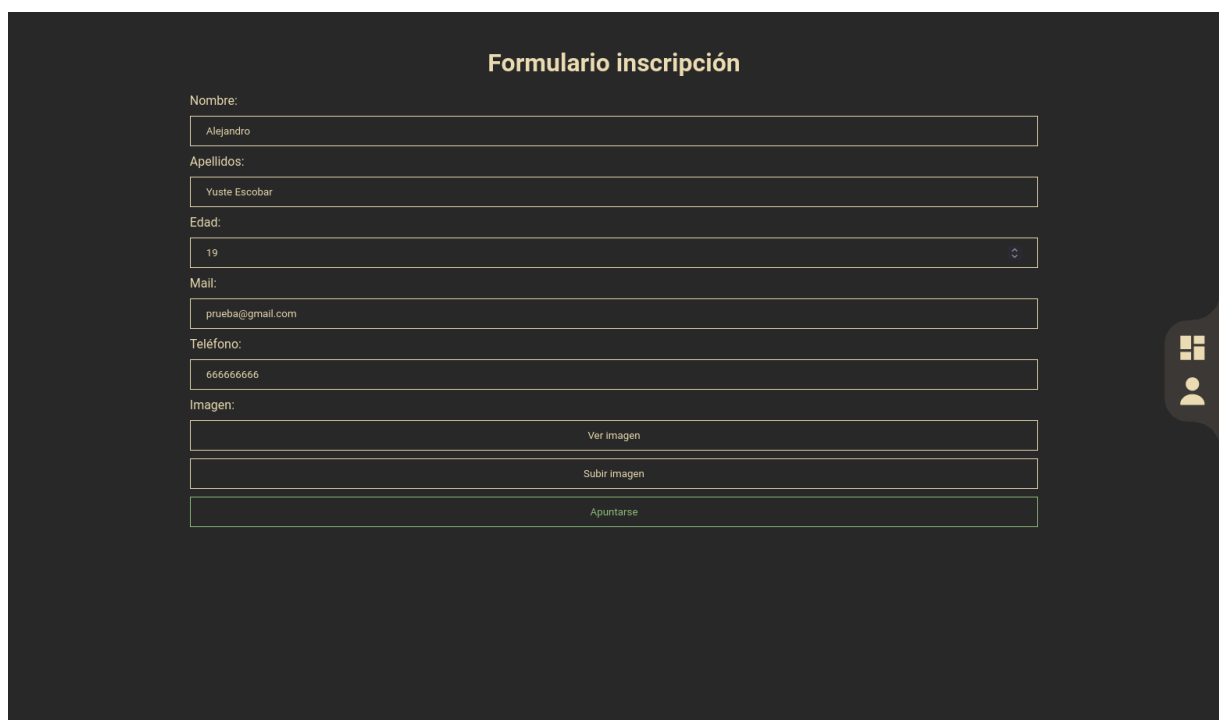
Edad:

Mail:

Teléfono:

Imagen:

Al rellenar los campos necesarios podremos inscribirnos haciendo clic en el botón de “Apuntarse”.



Formulario inscripción

Nombre:

Apellidos:

Edad:

Mail:

Teléfono:

Imagen:

Esperaremos hasta que se nos muestre una ventana confirmando que nuestro formulario fue enviado correctamente y podremos seguir utilizando la aplicación.

Formulario inscripción

Nombre:
Alejandro

Apellidos:
Yuste Escobar

Edad:
19

Mail:
prueba@gmail.com

Teléfono:
666666666

Imagen:
Ver imagen
Subir imagen
Apuntarse

Formulario enviado correctamente
Cerrar

La pantalla de perfil anteriormente mencionada nos muestra nuestros datos y la posibilidad de modificarlos, cambiar la contraseña y cerrar sesión, en caso de ser administrador también el acceso al panel de control.

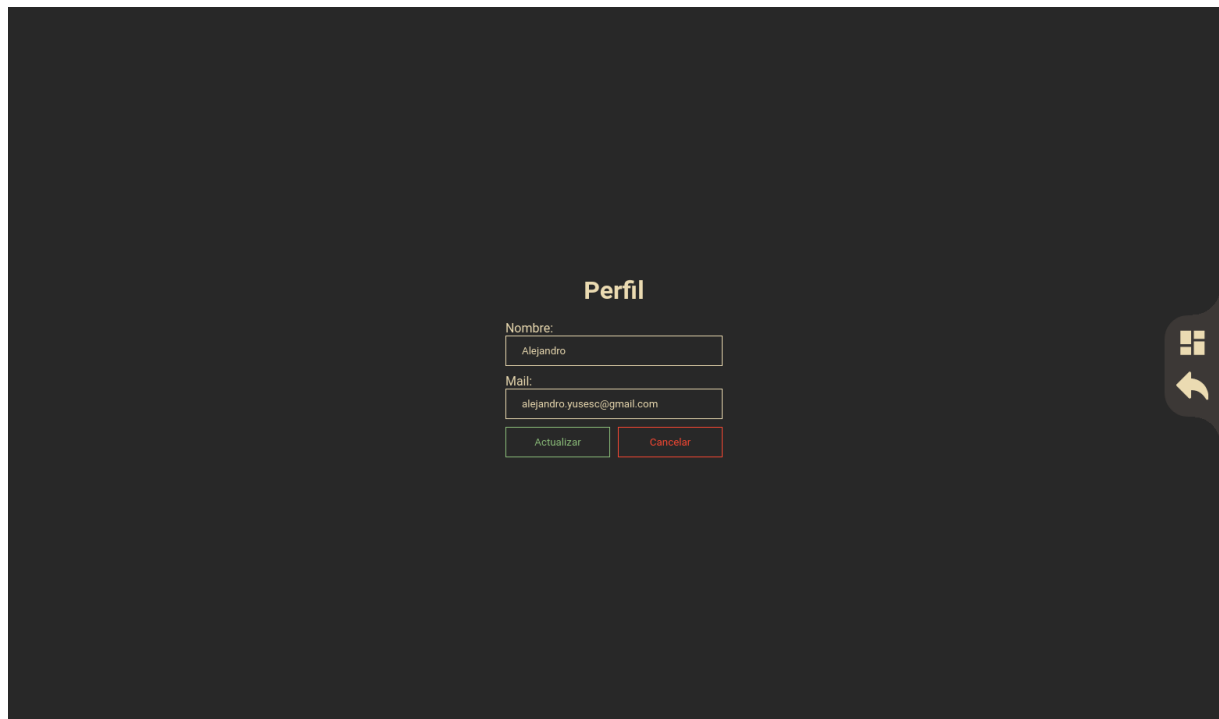
Perfil

Nombre:
Alejandro

Mail:
alejandro.yusesc@gmail.com

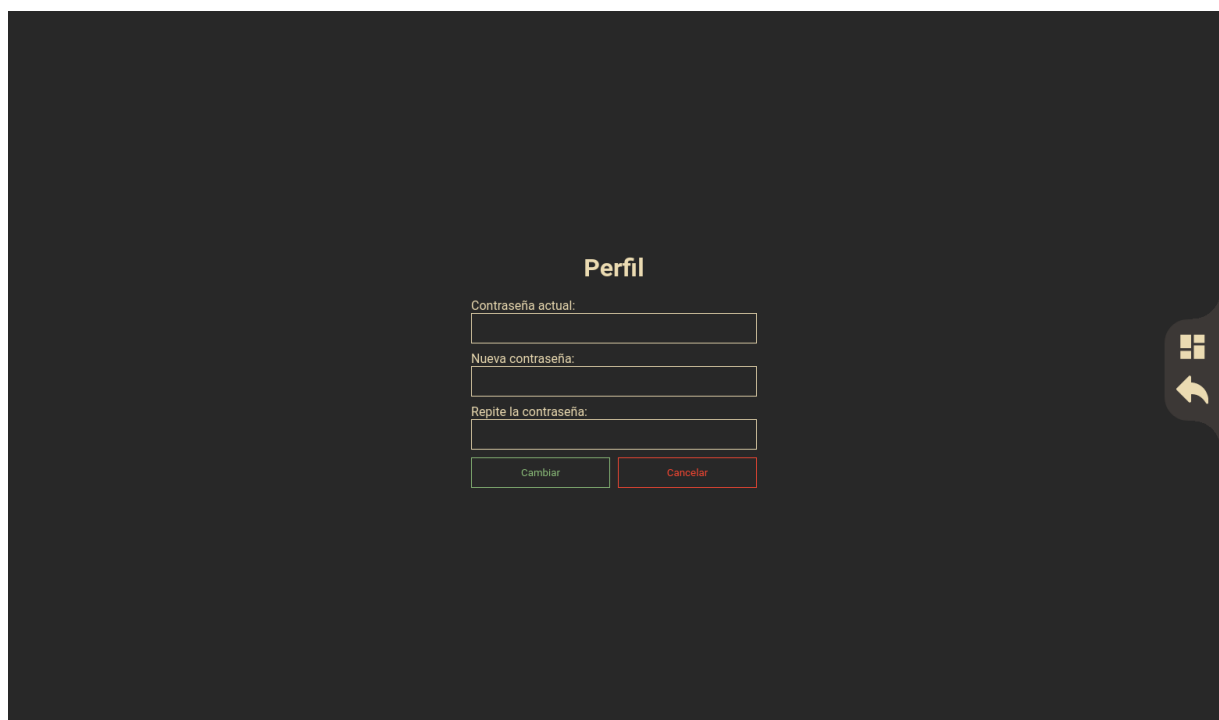
Modificar datos
Cambiar contraseña
Panel de control
Cerrar sesión

Al hacer clic en “Modificar datos” se nos mostrará el formulario con el que podremos cambiar nuestro nombre y/o mail.



The screenshot shows a dark-themed mobile application interface. At the top, the title 'Perfil' is centered in a light gray font. Below the title, there are two input fields. The first is labeled 'Nombre:' and contains the text 'Alejandro'. The second is labeled 'Mail:' and contains the text 'alejandro.yuseso@gmail.com'. Below these fields are two buttons: a green button labeled 'Actualizar' and a red button labeled 'Cancelar'. On the right side of the screen, there is a vertical sidebar with two icons: a grid icon at the top and a back arrow icon below it.

En caso de seleccionar “Cambiar contraseña” se nos pedirá la antigua contraseña más la nueva contraseña y repetir esta.



The screenshot shows the same dark-themed mobile application interface as the previous one. The title 'Perfil' is centered at the top. Below the title, there are three input fields. The first is labeled 'Contraseña actual:' and is empty. The second is labeled 'Nueva contraseña:' and is empty. The third is labeled 'Repite la contraseña:' and is empty. Below these fields are two buttons: a green button labeled 'Cambiar' and a red button labeled 'Cancelar'. On the right side of the screen, there is a vertical sidebar with two icons: a grid icon at the top and a back arrow icon below it.

En el caso de los administradores, al acceder al panel de control nos encontraremos con varias opciones de datos que podrán gestionar.



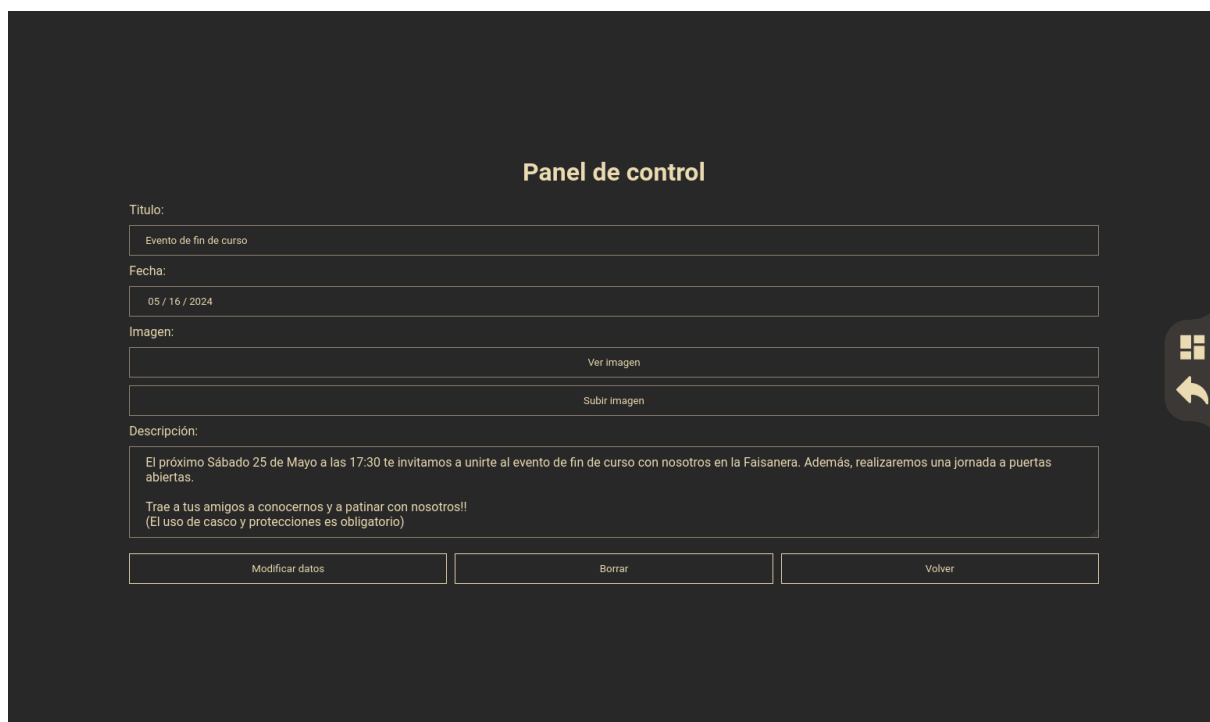
Al acceder a una de estas secciones podremos observar un listado de los datos con la opción de volver al menú en la parte superior izquierda, y un filtro en la parte derecha.



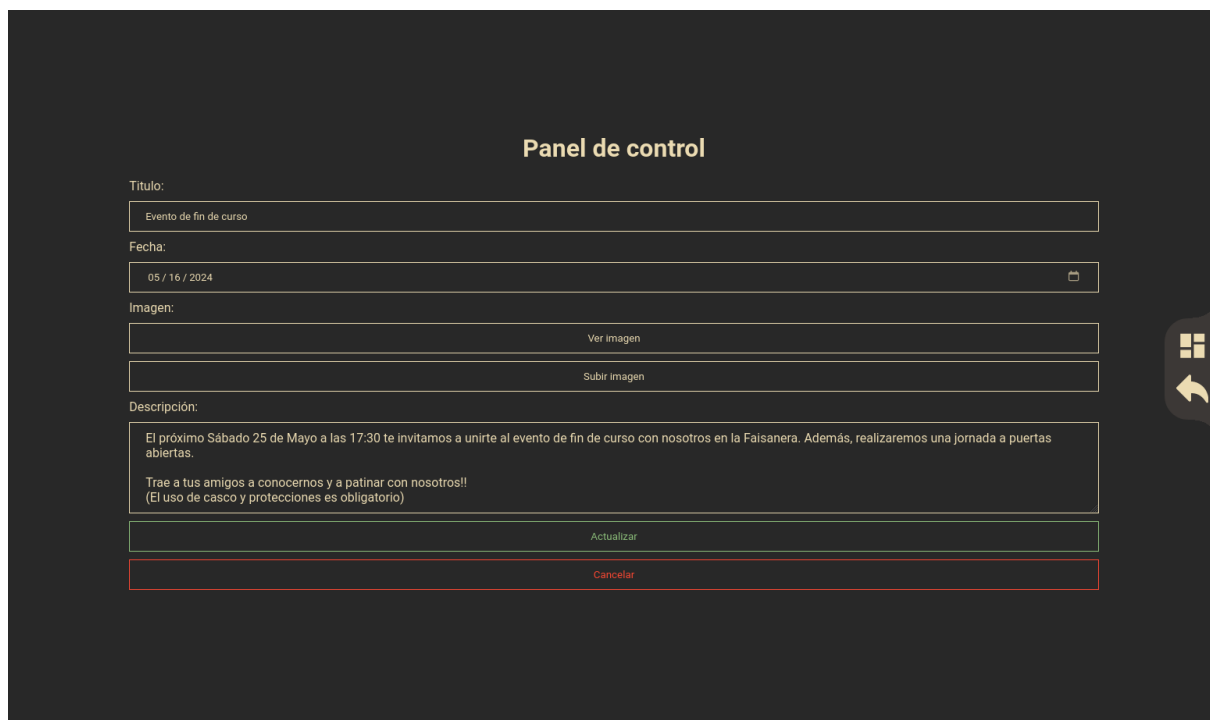
Este filtro es un campo de texto que filtrará entre títulos y contenidos de los datos.



Al hacer clic en uno de los registros podremos observar sus valores y las opciones de modificar los datos, borrar el registro o volver a la pantalla anterior.

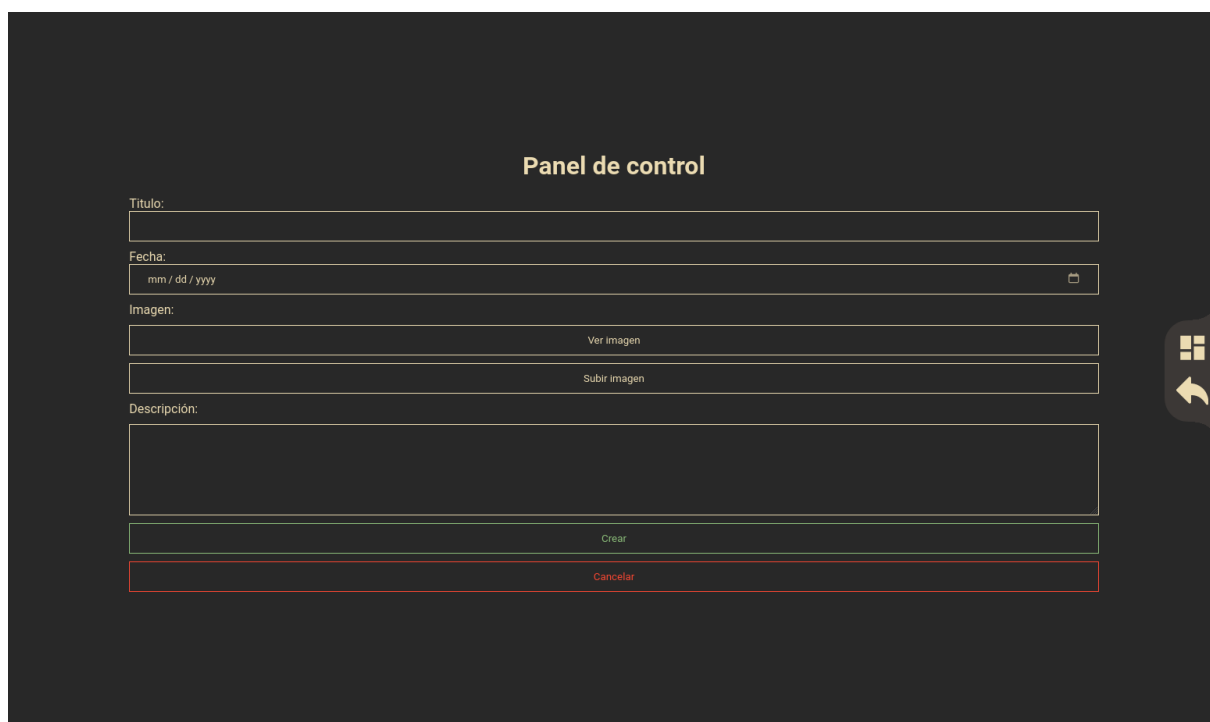


En caso de modificar los datos se nos presentará un formulario donde podremos actualizarlos o cancelar la operación.



The screenshot shows a dark-themed web interface titled "Panel de control". It contains a form for editing an event. The form fields are: "Titulo:" with the value "Evento de fin de curso"; "Fecha:" with the value "05 / 16 / 2024" and a calendar icon; "Imagen:" with two buttons, "Ver imagen" and "Subir imagen"; and "Descripción:" with the text "El próximo Sábado 25 de Mayo a las 17:30 te invitamos a unirme al evento de fin de curso con nosotros en la Faisanera. Además, realizaremos una jornada a puertas abiertas. Trae a tus amigos a conocernos y a patinar con nosotros!! (El uso de casco y protecciones es obligatorio)". At the bottom are two buttons: "Actualizar" (green) and "Cancelar" (red). On the right side, there is a vertical toolbar with a grid icon and a back arrow icon.

Además, en el caso de hacer clic en el botón con un “+” en la pantalla del listado de datos podremos crear un nuevo registro rellenando los datos del formulario o cancelar la operación.

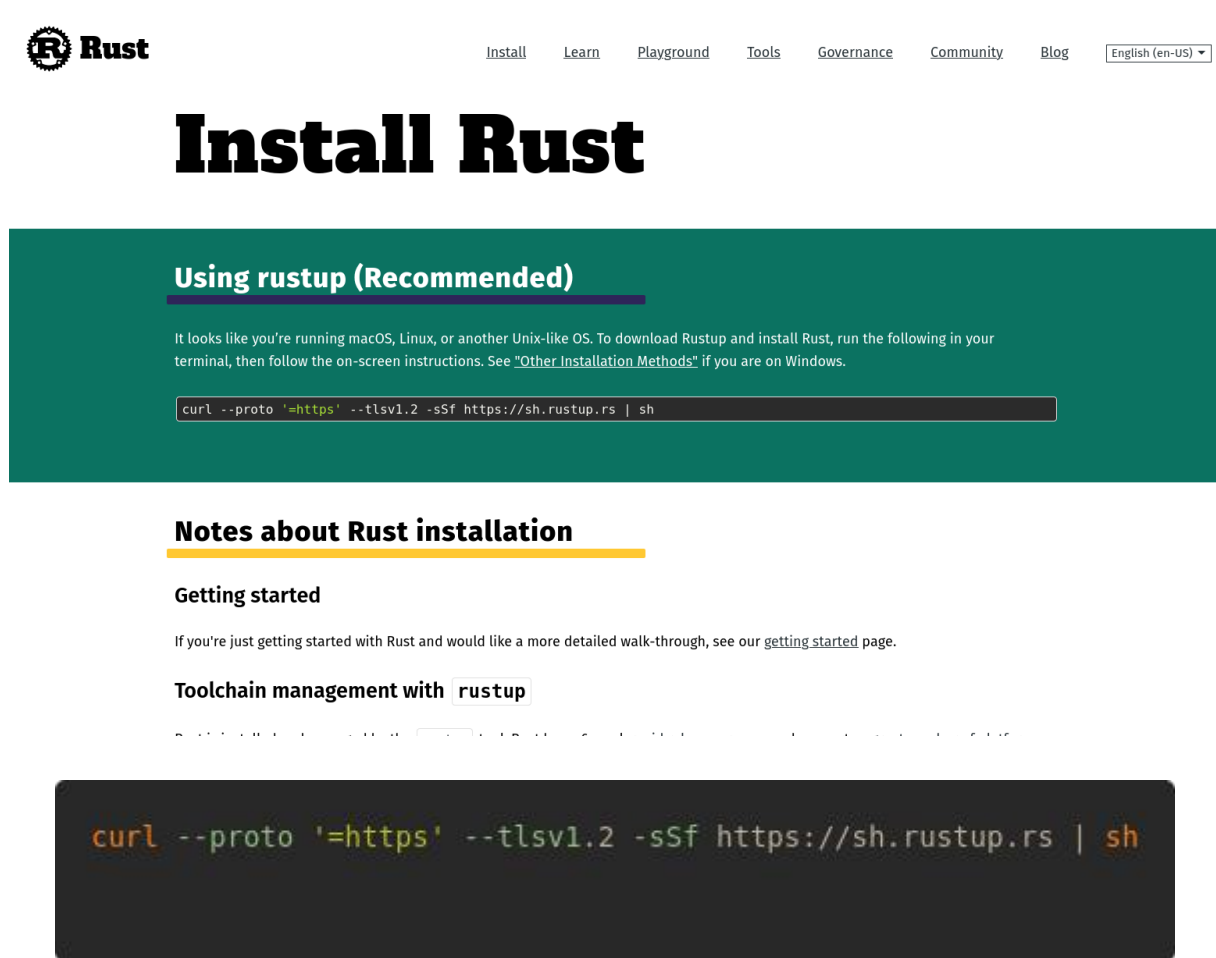


The screenshot shows the same dark-themed web interface titled "Panel de control", but in the "Crear" (Create) mode. The form fields are: "Titulo:" (empty); "Fecha:" with a placeholder "mm / dd / yyyy" and a calendar icon; "Imagen:" with two buttons, "Ver imagen" and "Subir imagen"; and "Descripción:" (empty). At the bottom are two buttons: "Crear" (green) and "Cancelar" (red). The right-side toolbar is identical to the previous screenshot, featuring a grid icon and a back arrow icon.

Manual de instalación

Ya que esta aplicación está desarrollada en rust deberemos instalar diferentes herramientas para manejar esto, además de las herramientas para correr docker.

Comenzando con las herramientas necesarias para correr y probar código rust necesitaremos instalar “rustup”, se puede instalar con el comando que nos ofrece rust en su página oficial o como se ajuste a tu sistema.

A screenshot of the Rust project's official installation page. At the top, there's a navigation bar with links for 'Install', 'Learn', 'Playground', 'Tools', 'Governance', 'Community', 'Blog', and a language dropdown set to 'English (en-US)'. The main heading is 'Install Rust'. Below it, a section titled 'Using rustup (Recommended)' provides instructions for macOS, Linux, and other Unix-like OSes. It includes a terminal command to install rustup. A 'Notes about Rust installation' section follows, with subsections for 'Getting started' and 'Toolchain management with rustup'. The 'Getting started' subsection mentions a 'getting started' page. The 'Toolchain management with rustup' subsection shows a terminal command to update rustup. The command is: `curl --proto 'https' --tlsv1.2 -sSf https://sh.rustup.rs | sh`.

Using rustup (Recommended)

It looks like you're running macOS, Linux, or another Unix-like OS. To download Rustup and install Rust, run the following in your terminal, then follow the on-screen instructions. See ["Other Installation Methods"](#) if you are on Windows.

```
curl --proto 'https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Notes about Rust installation

Getting started

If you're just getting started with Rust and would like a more detailed walk-through, see our [getting started](#) page.

Toolchain management with rustup

```
curl --proto 'https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Una vez instalado, con un simple comando podemos actualizar/installar nuestro inicio del entorno rust.

```
rustup update
```


Ahora podremos instalar nuestro cliente de terminal para SQLx con el sencillo comando que nos ofrece cargo al ser SQLx una librería de rust.

```
cargo install sqlx-cli
```

Con este cliente de terminal ahora podremos crear migraciones con el siguiente comando:

```
sqlx migrate add <nombre>
```

Pero antes de correrlas deberemos instalar docker para poder crear una base de datos de prueba. Esto se hará dependiendo de tu sistema, siendo los siguientes sistemas en los que docker está disponible.

The screenshot shows the 'Install Docker Engine' page on the Docker documentation website. The page is divided into several sections:

- Table of contents:** Includes links for 'Supported platforms', 'Other Linux distros', 'Release channels', 'Support', 'Upgrade path', 'Licensing', 'Reporting security issues', and 'Get started'.
- Supported platforms:** A table showing compatibility for various Linux distributions across different architectures.
- Other Linux distros:** A section for additional Linux distributions.

Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	ppc64le	s390x
CentOS	✓	✓		✓	
Debian	✓	✓	✓	✓	
Fedora	✓	✓		✓	
Raspberry Pi OS (32-bit)			✓		
RHEL (s390x)					✓
SLES					✓
Ubuntu	✓	✓	✓	✓	✓
Binaries	✓	✓	✓		

Una vez instalado docker podemos ejecutar un comando con el que correremos un contenedor con una base de datos en él:

```
docker run --name cochinillos-db \
  -e MYSQL_DATABASE=cochinillos \
  -e MYSQL_ROOT_PASSWORD=prueba \
  -p 3306:3306 -d mysql
```

Esto creará un contenedor con el nombre “cochinillos-db”, con una base de datos inicial “cochinillos”, con la contraseña del usuario root “prueba”, en el puerto 3306 y con la imagen oficial de mysql. Con lo que ahora si podremos ejecutar las migraciones asegurándonos que la configuración en nuestro archivo “.env” está correctamente, en nuestro caso:

```
DATABASE_URL=mysql://root:prueba@localhost:3306/cochinillos
MYSQL_PASSWORD=prueba
RUTA_UPLOADS="./uploads"
MAIL="Cochinillos Voladores <info@cochinillosvoladores.org>"
USUARIO_MAIL="info@cochinillosvoladores.org"
CONTRA_MAIL="Ieacnur224#"
```

El comando que nos proporciona SQLx para correr las migraciones es:

```
sqlx migrate run
```

Y por último, para hacer de la tarea del desarrollo algo más sencillo instalaremos la herramienta “cargo-watch” para que el código se recompile cada vez que hagamos un cambio a este, y ejecutaremos la herramienta.

```
cargo install cargo-watch
cargo watch -x run
```

En caso de querer parar la ejecución del código sencillamente se puede presionar “Ctrl+C” y en caso de querer parar la ejecución de nuestro contenedor de base de datos ejecutaremos el comando:

```
docker kill cochinillos-db
```

Para poder ejecutar la base de datos de nuevo podremos utilizar el comando:

```
docker start cochinillos-db
```

Y por último, para poder acceder a la base de datos de manera manual se podrá ejecutar el siguiente comando, en el caso de haber llamado el contenedor de la misma manera:

```
docker exec -it cochinillos-db mysql -p
```

Un útil comando para poder hacer copias de seguridad a partir de bases de datos MySQL desplegadas en un contenedor es la imagen de mysqldump oficial que podemos encontrar en docker:

```
docker exec cochinillos-db mysqldump -u root --password=prueba cochinillos > backup.sql
```

Siendo “cochinillos-db” el nombre del contenedor, “root” el nombre del usuario con el que accederemos, “prueba” la contraseña de ese usuario, “cochinillos” la base de datos que copiaremos y “backup.sql” al archivo en el que copiaremos los datos.

Conclusiones y posibles ampliaciones

Este proyecto me ha servido para poder aprender sobre cómo es desarrollar una aplicación desde cero, ya que nunca he tenido la oportunidad de experimentarlo. Hasta ahora no estaba del todo enterado de los problemas de las nuevas metodologías que se supone que están basadas en la metodología ágil y como llevar a cabo un desarrollo realmente acorde con esta metodología y lo mucho que se beneficia un proyecto de esta, durante todo el proceso el seguir la metodología ágil parecía lo más sensato y natural. Tampoco tenía la suficiente experiencia como para comparar con anteriores desarrollos y predecir unos buenos tiempos para cada sección del proyecto, esto sumado a la falta de conocimiento real de las herramientas dañó y retrasó bastante el proceso, aunque estoy bastante contento con haber conseguido superar estos problemas y de una forma que el resultado final sea algo funcional y ordenado. También, las tecnologías que he utilizado se han mostrado ser increíblemente útiles, sencillas y en general excelentes de utilizar pero también he notado que lenguajes como rust los cuales tienen un enfoque de más bajo nivel que otros lenguajes de servidor es que no compenentran de buena manera con la metodología ágil, ya que te pasas mucho tiempo preparando la base, aunque como se mencionó antes esto luego haga del proceso de desarrollo algo excepcional. Es por esto que quiero seguir probando lenguajes que se acomoden más a mis gustos y necesidades a la vez que se mantienen sencillos para poder servir al cliente con secciones funcionales lo antes posible, el lenguaje que estoy ahora estudiando es Gleam.

Algunas adiciones que me gustaría hacer a la aplicación es el dar más utilidad a los usuarios como puede ser un sistema de votación donde pueden expresar que es lo que creen que pasará en los siguientes partidos, como ejemplo una votación sobre si en el siguiente partido el equipo superará los 2 goles o cuantas expulsiones habrá en la primera parte. Además me gustaría implementar más pruebas de unidad ya que solo realicé unas de prueba para comprobar que el sistema funcionaba. También me gustaría mejorar el sistema de gestión de datos de los administradores ya que veo que actualmente es algo muy manual y posiblemente alguien que no tenga mucho entendimiento de su relación no pueda manejarlos de forma correcta.

Bibliografía

Let's Get Rusty *Deploy your Rust project in 20 minutes*. Disponible en:

https://www.youtube.com/watch?v=_gMzg77Qjm0

Jeremy Chone *rust-axum-course*. Disponible en: <https://github.com/jeremychone-channel/rust-axum-course>

Jeremy Chone *rust-web-app*. Disponible en: <https://github.com/rust10x/rust-web-app>

Rust foundation *rust*. Disponible en: <https://www.rust-lang.org/>

Mehcode y Abonander *SQLx crate*. Disponible en: <https://docs.rs/sqlx/latest/sqlx/>

Carllerche, Davidpdrsn y tokio-rs *Axum crate*. Disponible en:
<https://docs.rs/sqlx/latest/sqlx/>

Docker *Docker blog*. Disponible en: <https://www.docker.com/blog/>

ZeroSSL *Caddy*. Disponible en: <https://caddyserver.com/>

Deploy Rust App on VPS with GitHub Actions and Docker. Disponible en:
<https://codevoweb.com/deploy-rust-app-on-vps-with-github-actions-and-docker/>

Atlassian *Agile*. Disponible en: <https://www.atlassian.com/agile>

Atlassian *SCRUM*. Disponible en: <https://www.atlassian.com/es/agile/scrum>

No Boilerplate *You're doing agile wrong*. Disponible en:
<https://www.youtube.com/watch?v=9K20e7jIQPA>

Relación de ficheros en formato digital

- **memoria.pdf.** La memoria del proyecto.
- **README.md.** Documentaciones de las diferentes partes del repositorio.
- **.gitignore.** Archivo de texto para declarar los archivos que no queremos que se suban al repositorio.
- **.github/workflows/prod.yml.** El archivo del proceso de CI/CD anteriormente explicado.
- **bd.** En esta carpeta están todos los archivos relacionados con la base de datos.
- **funcionales.** En esta carpeta se encuentran ambos funcionales generados al inicio del proceso de la aplicación.
- **img.** En esta carpeta se encuentran todos los medios utilizados en la documentación o en la aplicación, imágenes y svgs.
- **codigo/cochinillos-voladores.** En esta carpeta se encuentra el código de la aplicación.
- **codigo/cochinillos-voladores/.dockerignore.** En este archivo declaramos los archivos que no queremos que docker utilice.
- **codigo/cochinillos-voladores/Caddyfile.** En este archivo configuramos el servidor de redirección “caddy”.
- **codigo/cochinillos-voladores/Cargo.lock.** Este es un archivo generado por cargo para guardar de forma declarativa la versión de todas las librerías utilizadas en la aplicación.
- **codigo/cochinillos-voladores/Cargo.toml.** En este archivo se declaran las librerías que utilizaremos en nuestro código de rust.
- **codigo/cochinillos-voladores/Dockerfile.** En este archivo se configura el proceso de creación del contenedor en el que se ejecuta nuestro código.
- **codigo/cochinillos-voladores/docker-compose.prod.yml.** En este archivo se configura el contenedor en el que se ejecutará el servidor de redirección.
- **codigo/cochinillos-voladores/docker-compose.yml.** En este archivo se configura los contenedores que se ejecutan para correr el código y la base de datos de la aplicación web.
- **codigo/cochinillos-voladores/.sqlx.** En esta carpeta se guardan todos los archivos que genera sqlx-cli al preparar la base de datos y el código.

- **codigo/cochinillos-voladores/assets.** En esta carpeta se guardan todos los archivos estáticos de la aplicación para que esta sea capaz de accederlos.
- **codigo/cochinillos-voladores/migrations.** En esta carpeta se guardan las migraciones creadas por la herramienta sqlx-cli para determinar la estructura de nuestra base de datos.
- **codigo/cochinillos-voladores/templates.** En esta carpeta se guardan todas las plantillas que utilizaremos mediante askama para servir el HTML al cliente. Se puede encontrar las páginas en la carpeta en si y componentes dentro de una carpeta llamada “componentes”.
- **codigo/cochinillos-voladores/src.** En esta carpeta se encuentra el código base de la aplicación escrito en rust.
- **codigo/cochinillos-voladores/src/main.rs.** Este es el primer archivo que ejecuta la aplicación gracias a contener la función con la derivación “tokio::main”.
- **codigo/cochinillos-voladores/src/error.rs.** Este es el archivo principal donde controlo todos los errores y donde guardo los generales. Montando así mi propio sistema de errores.
- **codigo/cochinillos-voladores/src/ctx.rs.** Archivo en el que se define la estructura del contexto de la aplicación.
- **codigo/cochinillos-voladores/src/mail.rs.** Archivo para manejar el mandar mails.
- **codigo/cochinillos-voladores/src/uploads.rs.** Archivo para manejar la subida de archivos por parte de los usuarios.
- **codigo/cochinillos-voladores/src/tests.** Carpeta donde se guardan todas las pruebas unitarias de código, siendo mod.rs la general y luego separándolo por cada entidad de nuestro modelo (solo hay dos pruebas a modo de demostración).
- **codigo/cochinillos-voladores/src/modelo.** Carpeta donde se guarda todo lo relacionado con nuestro modelo, teniendo incluso su propio error. En esta carpeta se encuentra un archivo por cada entidad de nuestro modelo.
- **codigo/cochinillos-voladores/src/web.** Carpeta donde se guarda las rutas de la aplicación y todo lo relacionado con peticiones.
- **codigo/cochinillos-voladores/src/web/res_map.rs.** Archivo que controla todas las respuestas que envía la aplicación añadiendo algo de lógica.

- **codigo/cochinillos-voladores/src/web/api.** Las rutas relacionadas con la conexión con el modelo.
- **codigo/cochinillos-voladores/src/web/auth.** Archivos para controlar la autenticación en la aplicación mediante middleware
- **codigo/cochinillos-voladores/src/web/paginas.** Archivos relacionados con las rutas de las diferentes páginas de la aplicación.

Glosario

- **caddy.** Servidor web moderno y fácil de usar que se destaca por su configuración automática de HTTPS utilizando “Let's Encrypt”. Es conocido por su simplicidad, configuración en formato texto y características avanzadas como la gestión automática de certificados SSL/TLS, soporte para HTTP/2, y capacidades de proxy inverso. Caddy es especialmente popular para aplicaciones web y entornos de desarrollo debido a su facilidad de configuración y su enfoque en la seguridad.
- **Cadena de conexión.** Cadena de texto que contiene la información necesaria para que una aplicación se conecte a una base de datos específica. Por lo general, incluye detalles como el tipo de base de datos (MySQL, PostgreSQL, etc.), la dirección del servidor, el puerto, el nombre de la base de datos y las credenciales de autenticación (nombre de usuario y contraseña). Esta cadena se utiliza para establecer la conexión entre la aplicación y la base de datos, permitiendo a la aplicación acceder y manipular los datos almacenados en ella.
- **cargo.** Administrador de paquetes y construcción de proyectos para Rust. Permite crear, compilar y gestionar proyectos Rust de manera sencilla. Además de gestionar las dependencias del proyecto, Cargo automatiza tareas como la compilación, la ejecución de pruebas y la generación de documentación. También facilita la distribución de paquetes de Rust a través del registro de paquetes crates.io. En resumen, Cargo simplifica el desarrollo de software en Rust al proporcionar una herramienta unificada para administrar todo el ciclo de vida del proyecto.
- **Código abierto.** software cuyo código fuente es libremente accesible, modificable y distribuible por cualquier persona. Este tipo de software fomenta

la colaboración y la transparencia, permitiendo a desarrolladores y usuarios mejorar el software, corregir errores y adaptar las aplicaciones a sus necesidades específicas. Ejemplos populares de software de código abierto incluyen Linux, Git, y Firefox.

- **Contenedor.** unidad de software que empaqueta una aplicación y sus dependencias en un entorno aislado y portátil. Utiliza las capacidades del sistema operativo subyacente para ejecutar aplicaciones de manera consistente en cualquier entorno, independientemente de las diferencias entre las plataformas donde se ejecuta. Los contenedores son ligeros y eficientes, ya que comparten el kernel del sistema operativo, a diferencia de las máquinas virtuales que incluyen su propio sistema operativo completo. Herramientas populares para gestionar contenedores incluyen Docker y Kubernetes.
- **Dominio.** Dirección web única que se utiliza para identificar un sitio web en Internet. Está compuesto por un nombre y una extensión (como .com, .org, .net). Por ejemplo, en "cochinillosvoladores.org", "cochinillosvoladores" es el nombre del dominio y ".org" es la extensión. Los dominios son más fáciles de recordar que las direcciones IP numéricas y son gestionados por registradores de dominios. Cuando un usuario ingresa un dominio en su navegador, el sistema de nombres de dominio (DNS) traduce ese nombre a la dirección IP del servidor donde está alojado el sitio web, permitiendo el acceso al contenido del mismo.
- **Entrega continua (CD).** Extensión de CI que automatiza la entrega del código integrado a un entorno de producción o pre-producción, asegurando que esté siempre en un estado desplegable.
- **Gestor de paquetes.** Herramienta que automatiza la instalación, actualización, configuración y eliminación de software en un sistema operativo. Maneja paquetes, que son colecciones de archivos necesarios para un software específico, y resuelve las dependencias entre ellos para asegurar que todo funcione correctamente. Como por ejemplo apt para Ubuntu/Debian.
- **Imagen (contenedor).** Paquete de solo lectura que incluye todo lo necesario para ejecutar una aplicación: el código, las bibliotecas, las dependencias y las

configuraciones necesarias. Las imágenes sirven como una plantilla para crear contenedores. Una vez que una imagen está disponible, se puede usar para lanzar múltiples contenedores en cualquier entorno compatible. Las imágenes son gestionadas y distribuidas a través de registros de imágenes, como Docker Hub.

- **Integración continua (CI).** Práctica de desarrollo de software donde los desarrolladores integran su código frecuentemente en un repositorio compartido, realizando pruebas automáticas para detectar errores rápidamente.
- **Modelo Vista Controlador (MVC).** Patrón de diseño de software que divide una aplicación en tres componentes interconectados: el Modelo, que maneja los datos y la lógica de negocio; la Vista, que presenta la interfaz de usuario y muestra los datos; y el Controlador, que actúa como intermediario, procesando las entradas del usuario, actualizando el modelo y determinando qué vista mostrar. Esta separación de responsabilidades facilita la organización del código, mejora la escalabilidad y simplifica el mantenimiento y las pruebas de la aplicación.
- **push.** Comando de Git que se utiliza para subir los cambios locales de un repositorio a un repositorio remoto. Esto actualiza el repositorio remoto con los commits que se han realizado en el repositorio local.
- **Repositorio.** Lugar de almacenamiento donde se guarda y gestiona el código fuente de un proyecto, junto con su historial de cambios. Permite a los desarrolladores colaborar, realizar seguimientos de versiones y gestionar diferentes ramas del desarrollo del software.

- **SCRUM.** Marco ágil para gestionar proyectos, especialmente en desarrollo de software. Funciona en ciclos cortos llamados sprints, que duran entre una y cuatro semanas, y se enfoca en la entrega incremental de productos. En Scrum, hay roles específicos como el Scrum Master, el Product Owner y el Equipo de Desarrollo. Se realizan reuniones regulares para planificar, revisar el progreso diario, evaluar el trabajo completado y reflexionar sobre el proceso para mejorarlo continuamente. Scrum promueve la colaboración, la flexibilidad y la entrega constante de valor.
- **Servidor virtual privado (VPS).** Servicio de hosting que proporciona recursos dedicados en un servidor compartido. Utiliza tecnología de virtualización para dividir un servidor físico en varios servidores virtuales independientes, cada uno con su propio sistema operativo, almacenamiento y recursos de cómputo. Un VPS ofrece mayor control, flexibilidad y rendimiento en comparación con el hosting compartido, y es una opción popular para sitios web, aplicaciones y servicios en línea que requieren un entorno más aislado y configurable.
- **UUID.** Identificador único de 128 bits utilizado para identificar de manera única información en sistemas distribuidos. Se representan como una cadena de 32 caracteres hexadecimales, normalmente dividida en cinco grupos separados por guiones (por ejemplo, 123e4567-e89b-12d3-a456-426614174000). Los UUID son diseñados para ser únicos a través del tiempo y el espacio, lo que los hace ideales para usos como claves en bases de datos, identificadores de objetos en sistemas distribuidos y etiquetas únicas para recursos.