Institut Puig Castellar
Santa Coloma de Gramenet

Daemon

Projecte de desenvolupament

CFGS Administració de Sistemes Informàtics i Xarxes

Autors
Grup

Resum del projecte (màxim 250 paraules):

Este proyecto tiene como temática principal la creación y el despliegue de servidores de Luanti *(antes conocido como Minetest)*, un videojuego de código abierto que es similar a Minecraft, aunque distanciándose por mucho en cuanto a mecánicas y manera de juego se trata. El objetivo es facilitar la gestión de este mediante una interfaz web desarrollada en PHP, conectada a una base de datos que usará PostgresSQL y que se comunicarán por una API desarrollada en Python.

Se tratará de proporcionar una solución auto alojada, accesible y flexible para administrar servidores de Luanti, en general orientado al aprendizaje de tecnologías libres. La idea central de todo esto es el poder controlar y automatizar varias configuraciones del servidor *(ejemplos pueden ser los mods, usuarios o algunos ajustes del mundo)* desde una interfaz sencilla y entendible, especialmente útil para cualquier usuario.

La metodología seguida hasta el momento ha sido práctica y modular *(iterativa incremental)*: se comenzó instalando y configurando Lunati en un contenedor *(utilizando los contenedores Podman)*, luego mediante el uso de bash, se conectó a una base de datos PostgresSQL para tener un recuento de los servidores creados, y finalmente utilizando HTML y PHP se desarrolló un entorno web para que los usuarios puedan ver los mundos que tienen creados y poder interactuar con ellos, ajustando configuraciones o eliminado o creando nuevos.

Como conclusión, el proyecto intenta implementar un sistema completo de gestión de servidores de Luanti de manera modular, segura y escalable aprovechando tecnologías de software libre.

Paraules clau (entre 4 i 8):

- Luanti
- Servidor
- PHP
- PostgreSQL
- Software libre
- Contenedores
- Automatización
- Python

Debido a que el PDF pesa 8MB por las imágenes y solo se puede subir un archivo de 5MB, comparto un link con el PDF sin comprimir con las imágenes con un poco más de calidad. El PDF estará en mi Drive → Daemon Projecte ASIX.pdf.

Índex

Llista de figures

1 Introducció

Este proyecto tiene como objetivo el diseño e implementación de un sistema de gestión de servidores Luanti, un videojuego de mundo abierto y libre, que está orientado principalmente a la experimentación educativa y al aprendizaje de tecnologías abiertas así como también alentando a sus usuarios a compartir sus creaciones. Permitiendo a cualquier persona, sin necesidad de conocimientos técnicos avanzados, pueda crear, personalizar con cada nueva actualización que tiene integrando más programas, y administrar su propio mundo de juego. La finalidad del proyecto es el hospedar un servidor o mundo para cualquier persona que quiera algo rápido y sencillo, sin necesidad de tener que leer guías, inspirandose en sitios como Aternos.

La propuesta parte del interés por ofrecer un entorno que combine el potencial lúdico de los videojuegos con herramientas modernas de infraestructura, tales como contenedores *(Podman)*, bases de datos *(PostgreSQL)* y desarrollo web *(HTML y PHP)*, buscando en lo más que se pueda, el automatizar procesos que normalmente requerirían intervención técnica manual. Asimismo simplificar la puesta en marcha de servidores personalizados sin necesidad de conocimientos profundos.

1.1 Context

Actualmente, Luanti es una plataforma ciertamente muy utilizada en entornos educativos y comunidades de software libre debido a la flexibilidad y bajo consumo de recursos con el cual fue programada. Sin embargo, el llegar a montar un servidor propio aún implica ciertos conocimientos técnicos como la gestión de puertos, configuración de mundos *(incluyendo mods), tocar, agregar, o eliminar líneas en ficheros* y el uso de línea de comandos. Este proyecto pretende acortar esa brecha o cerrarla por completo proporcionando un entorno que automatice estas tareas lo más que se pueda, permitiendo que cualquier persona o usuario pueda crear su propio mundo con solo unos pocos clics.

1.2 Justificació

El presente proyecto surge no como una idea inicial ya pensada con anterioridad, sino que era una de las propuestas de proyectos que se me presentaron. Y no lo tomé porque era la única opción, sino porque genuinamente me llamó la atención la idea de realizar todo ese proceso y ver que tanto puedo aprender de ello. En la investigación se ve la necesidad de conectar con una herramienta accesible y automatizada para el despliegue de servidores de Luanti, especialmente en entornos comunitarios o a niveles personales. A pesar de que existen herramientas generales para la creación de servidores, muchas de ellas requieren algunos conocimientos técnicos. Y en general, los tutoriales que se tienen, piden hacer uso de una terminal o configuraciones manuales que puede llegar a ser una barrera para usuarios con poca experiencia.

En este contexto, el sistema propuesto tiene una doble relevancia:

- Por un lado, facilitar el acceso a la gestión de servidores mediante una interfaz web sencilla, no muy rebuscada, y clara, reduciendo la complejidad técnica asociada a tareas como la configuración de volúmenes o el despliegue de contenedores *(que sí algo pesado y lioso en Luanti)*.

- Por otro lado, fomentar el aprendizaje en el ámbito de la informática, al integrar tecnologías como Podman, PostgreSQL y PHP en un proyecto práctico, funcional y con aplicación directa. Que si bien este no es el motivo principal, si alguien aprende de ello, qué mejor.

Además que, a raíz de querer facilitar lo más que se pueda, surge la necesidad de automatizar procesos mediante scripts que ayuden a reducir lo más que se puedan los errores humanos y asegurar un entorno más controlado.

Cómo ya mencioné, la herramienta busca ser lo más optimizada posible en ámbitos tanto técnicos como también en servicio al usuario. Como varios han sido acostumbrados a la interfaz de Aternos, hacer que si se ocupa la herramienta, no se sienta un salto o un cambio enorme, sino que familiar hasta cierto punto.

1.3 Objectius

1.3.1 Objectiu general

Diseñar e implementar un sistema automatizado para la creación y gestión de servidores del videojuego Luanti, accesible a los mundos creados por una interfaz web y sustentado por una base de datos relacional, contenedores y scripts personalizados.

1.3.2 Objectius específics

- Automatizar el despliegue de servidores Luanti, estos siendo en su versión contenedor *(Docker, pero usaré Podman)*.
- Permitir que cada usuario pueda crear y gestionar sus propios mundos de forma aislada y segura.
- Integrar una base de datos PostgreSQL para registrar los mundos y sus respectivos dueños.
- Desarrollar una interfaz web en PHP para facilitar el acceso al usuario a la vista de sus mundos, sin necesidad de usar el terminal.
- Establecer, en la medida de lo posible, una estructura de carpetas organizadas por usuarios y mundos, para el almacenamiento local *(en el servidor en este caso)*.
- Asignar puertos automáticamente a cada servidor *(contenedor)* desplegado para evitar conflictos.

1.4 Estratègia i planificació del projecte

Para llevar a cabo este proyecto se optó por desarrollar un sistema inspirado en algunos hostings ya posicionados en el mercado, pero adaptado a las necesidades de un entorno reducido: el ecosistema de Luanti. Nuevamente, aunque existen herramientas para desplegar servidores, ninguna está orientada a la experiencia de usuario desde un entorno comunitario con mínima intervención técnica.

La estrategia consiste en construir desde cero los elementos esenciales *(o lo que se tiene previsto que se usarán)*: scripts de automatización, el diseño de una base de

datos y una interfaz web funcional. Todo ello implementándolo de manera modular, con foco en la escalabilidad y en la integración fluida entre los componentes.

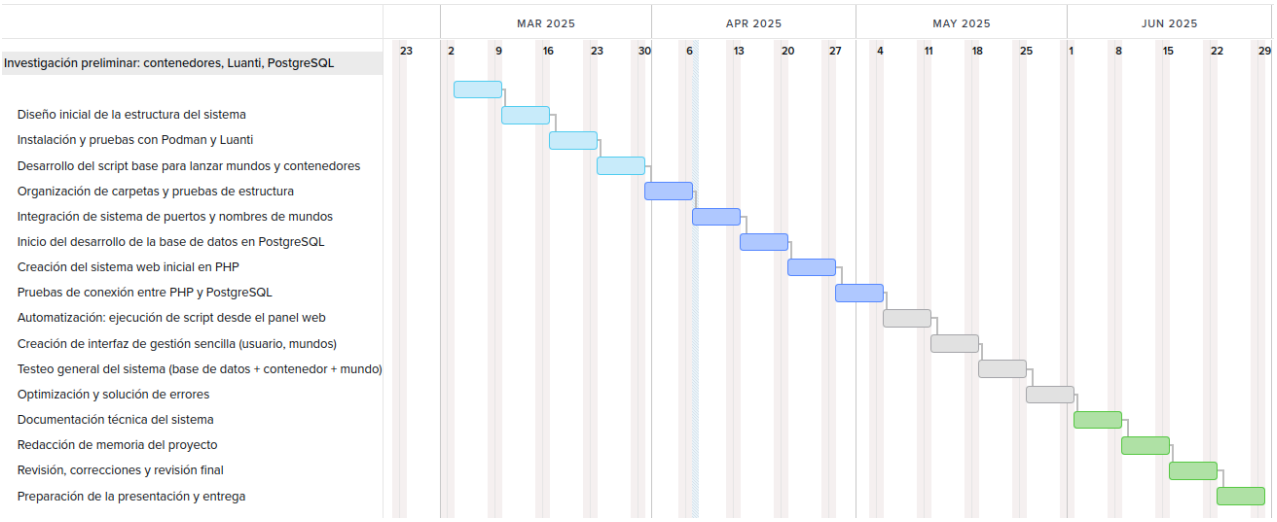
Se valora esta estrategia como viable, dado que se aprovechan tecnologías maduras y bien documentadas como Podman, PostgreSQL y el propio Luanti, lo cual permite centrar los esfuerzos en la lógica del sistema sin necesidad de desarrollar desde cero motores de juego o sistemas de contenedores.

1.5 Metodologia de treball

Se seguirá una metodología iterativa incremental, desarrollando el sistema por etapas con pruebas continuas y mejoras progresivas. Esta metodología es adecuada ya que permite adaptarse a cambios o descubrimientos técnicos que se presenten durante el desarrollo del proyecto *(lo cual sería clave más adelante)*, a la vez que ofrece resultados parciales funcionales en cada iteración.

Para la planificación se empleará un diagrama de Gantt para el desarrollo a lo largo del tiempo, intentando cumplir lo más posible las fechas establecidas.

En general, será tomado un enfoque semanal para distintas tareas, por lo que el plano queda hasta ambiguo:



1.6 Estudi econòmic i pressupostari

El desarrollo del proyecto principalmente está pensado para realizarse con software libre y recursos propios, algo antiguos se ha de mencionar, por lo tanto el coste directo de licencias o plataformas es nulo. Se podría considerar que el presupuesto se enfoca más en el tiempo invertido por el desarrollador y el uso de equipos personales, aunque en su desarrollo inicial se hace uso de máquinas virtuales.

Inventario de recursos y materiales:

- Ordenador portátil con sistema operativo Linux *(ya disponible)*. Y como mencioné anteriormente, se usó una máquina virtual para el desarrollo.

- **Herramientas de desarrollo**: editores de texto, servidor local, navegador *(software libre)*.
- **Tecnologías empleadas**: Podman, PostgreSQL, Minetest, PHP.

Coste estimado por horas de trabajo:

- Marzo: planificación y diseño *(15 horas)*.
- Abril: implementación de scripts y automatizaciones *(20 horas)*.
- Mayo: diseño e implementación de base de datos e interfaz web *(30 horas)*.
- Junio: pruebas finales, corrección de errores y redacción de la documentación *(20 horas)*.

Total estimado: 85 horas de trabajo. Podríamos decir que unas 100 si redondeamos.

Mantenimiento: El sistema puede mantenerse con bajo coste gracias a la modularidad y documentación del proyecto. El proyecto también se está desarrollando con la finalidad de consumir lo menos posible.

Beneficios y oportunidades:

- Posible uso en centros educativos, un hackathon *(les sería muy fácil)* o servidores comunitarios.
- Base para futuros desarrollos más avanzados como control de acceso, edición colaborativa o la ampliación a otros juegos o programas.

2 Descripció del projecte

2.1.1 Anàlisi de requisits

Para poder considerar funcional al sistema debe cumplir con los siguientes requisitos:

Funcionales:

- El usuario debe de poder crear, iniciar y detener servidores de Luanti desde una interfaz web.
- Cada servidor debe estar contenido de forma aislada utilizando Podman.
- El sistema debe poder almacenar información de los servidores y usuario en una base de datos PostgreSQL.
- El sistema debe permitir a los usuarios conocer el estado de sus servidores.
- Seguridad básica para la prevención de ejecuciones no autorizadas.
- Robustez frente a errores comunes, un ejemplo es la creación de servidores duplicados.
- Facilidad de uso y acceso a través de un navegador sin depender del terminal.

- La interfaz lo más clara y simple posible para usuario con pocos conocimientos técnicos.

Requisitos mínimos del sistema:

- **Hardware**: procesador de 2 núcleos, 4GB RAM o 6-8 GB si se usa escritorio y navegador, 20GB almacenamiento si es posible que sea un SSD *(no recomiendo pendrives)*.
- **Red**: conectividad local o acceso a red con NAT/PAT si se desea acceso externo.
- **Sistema operativo**: Linux *(preferiblemente un Ubuntu server o similar)*, con soporte a Podman.
- **Aplicaciones necesarias**: Podman, PostgreSQL, Apache2 y el módulo php, PHP, Python y el microframework Flask.

2.2 Tecnologies

2.2.1 Comparativa de les tecnologies valorades

Docker vs Podman: Docker no se tuvo muy en cuenta al inicio, sino que se fue directo con Podman, pero se menciona porque la mayoría de contenedores *(incluso el de Luanti)* están hechos para ejecutarse con Docker. Docker requiere un demonio en ejecución y privilegios de root, pero se optó por Podman al ser rootless y más seguro en entornos educativos. Y la mayoría de contenedores que requieren Docker se pueden ejecutar con Podman.

MySQL vs PostgreSQL: Lo mismo que Docker, pero este si se consideró un poco más de tiempo, se dejó en PostgreSQL al permitir máyor control y tipos de datos avanzados *(que no es que se use mucho este último)*, ideal para la gestión de mundos y usuarios. Además que está diseñado para manejar grandes volúmenes de datos.

PHP vs Node.js: Node.js a mi parecer no lo iba a necesitar, solo se consideró durante la investigación, como por un par de minutos, pero luego se optó por PHP que es más “sencillo” para implementar rápidamente formularios y lógica de servidor sin dependencias adicionales. Aunque javascript si se tiene en cuenta para detalles de la página.

Ubuntu Desktop vs Ubuntu Server: La decisión se consideró junto con MySQL y el querer hacer algo llamativo a nivel visual, pero con la visión de querer hacer que no se consuma la mayor cantidad de recursos posibles hizo que se optará por Ubuntu server.

2.2.2 Tecnologies escollides

Podman: Fue elegido de primero por, de nuevo, su enfoque sin daemon y mejor integración en servidores con políticas de seguridad restrictivas. Consume menos recursos al no requerir un demonio. Seguridad al no requerir de permisos elevados. E individualidad, ya que al no centralizarse en un demonio, estos son independientes totalmente los unos de los otros, incluso si el contenedor padre falla.

PostgreSQL: Por su mayor robustez y confiabilidad, mejor rendimiento en escrituras frecuentes y la seguridad en la integridad de datos. Aparte que, PostgreSQL está más orientado a la línea de comandos.

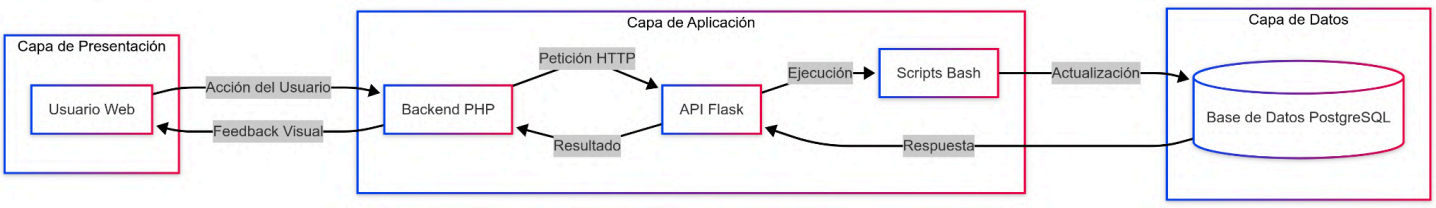
PHP: Por su "facilidad" de uso y amplia disponibilidad de recursos educativos y servidores ligeros. Adaptable al lenguaje HTML y que permite la automatización de tareas.

Luanti: Elegido por su enfoque Open Source, su naturaleza libre y personalizable. Ofrece el hostear un servidor en un contenedor y al ser tan personalizable, es más flexible al momento de dejar que el usuario diseñe su mundo.

2.3 Estructura del projecte

El sistema se compone de cuatro elementos principales:

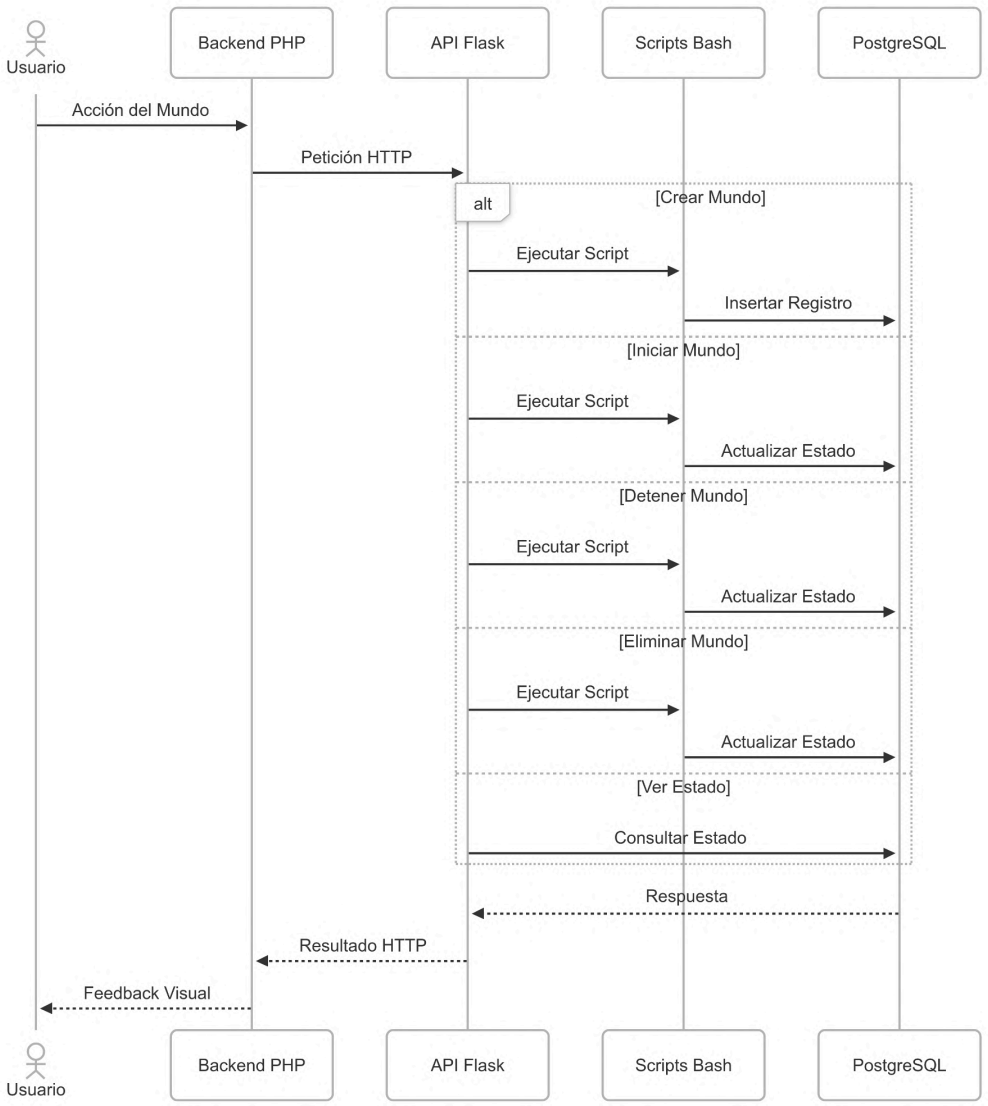
1. **Interfaz web *(PHP)***: Punto de interacción principal para el usuario. Desde aquí se pueden crear, iniciar, detener y eliminar mundos del servidor.
2. **API local *(Flask en Python)***: Actúa como intermediario entre la web y los scripts del sistema. Se encarga de recibir peticiones y ejecutar los scripts correspondientes de forma controlada.
3. **Sistema de contenedores *(Podman)***: Responsable de instanciar, detener y eliminar contenedores con el juego Luanti, encapsulando los mundos creados por cada usuario.
4. **Base de datos *(PostgreSQL)***: Almacena información sobre los usuarios, los mundos creados, su estado y el puerto asignado.



La comunicación entre estos componentes sigue este flujo:

- El usuario realiza una acción desde la web (por ejemplo, crear un mundo). ↴
- El backend en PHP realiza una petición HTTP a la API Flask, enviando los datos necesarios *(usuario, mundo, plantilla...)*. ↴
- La API ejecuta un script en Bash que gestiona la acción solicitada *(crear mundo, iniciar contenedor, detener, etc.)*. ↴
- Si corresponde, el script actualiza la base de datos PostgreSQL *(por ejemplo, registrando el nuevo mundo)*.

- Finalmente, el usuario recibe una respuesta visual (éxito/error) y se actualiza la información mostrada en la web.



2.4 Descripció dels components

- **Frontend web *(PHP)*:**

  Interfaz visible para el usuario. Genera formularios para crear, iniciar, detener y eliminar servidores. También presenta mensajes de confirmación o error según la acción realizada. Toda la interacción se hace sin necesidad de usar la terminal, lo que facilita su uso para personas sin conocimientos técnicos.

- **API local *(Flask)*:**

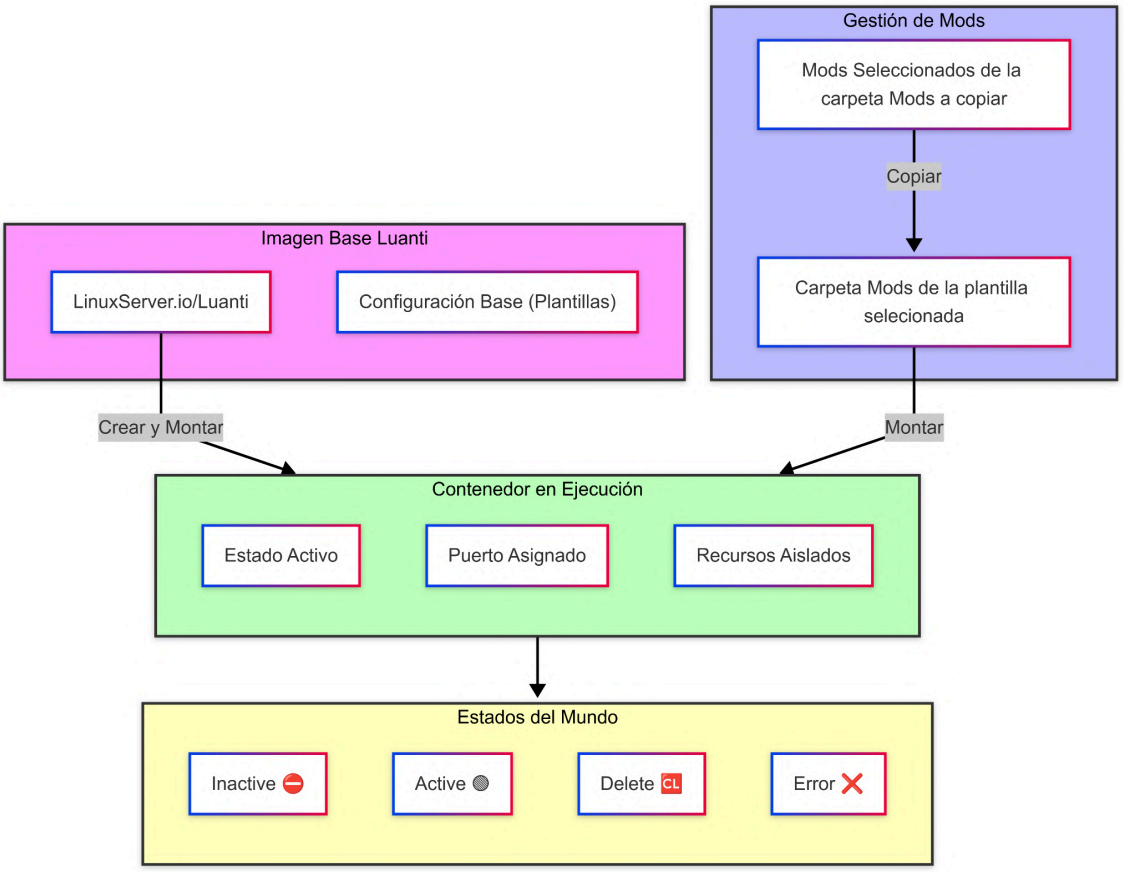
  Servidor intermedio que expone varios endpoints *(/start, /stop. /init, /delete, /status, /add_mods)* que se usa para gestionar las operaciones sobre los mundos. Es quién recibe las peticiones del frontend y ejecuta los scripts de shell correspondientes, lo que asegura una clara separación entre la lógica web y la del sistema.

- **Backend *(Scripts Bash):***

  Scripts encargados de ejecutar acciones específicas que se definieron previamente, como la de crear mundos, copiar mods. iniciar contenedores, detenerlos o eliminarlos. También son responsables de registrar los cambios en la base de datos. Están diseñados para ser reutilizables desde la API y facilitar la automatización.

- **Sistema de contenedores *(Podman)*:**

  Utilizado para lanzar instancias del juego Luanti. Cada mundo se ejecuta en su propio contenedor de forma aislada. Se emplea la imagen oficial de Luanti que está disponible en LinuxServer.io. Para mundos con mods, los scripts, utilizados solo dos en este proceso, copian los mods seleccionados a la carpeta correspondiente antes de lanzar el contenedor, que es donde se emplea el segundo script que mencioné anteriormente.



- **Base de datos *(PostgreSQL)*:**

  Gestiona toda la información relacionada con usuarios, mundos, puertos asignados, estado de cada servidor, etc. Permite por lo general consultar rápidamente que mundos existen, plantillas usadas para aquellos mundos, y las acciones mencionadas en el apartado Backend.

2.5 Definició de les funcionalitats

A continuación se detallan todas las funcionalidades principales que ofrece la solución implementada. Se explica el objetivo de cada una, el proceso conceptual para llevarla a cabo y su estado de implementación actual:

Creación de servidores Luanti

Permite a los usuarios generar un servidor seleccionando una plantilla predeterminada *(juego base o otros)* o añadiendo mods personalizados. La creación se gestiona desde la web mediante un formulario que envía los datos a una API local, que llama a un script en bash que prepara un entorno, copia la plantilla, añade los mods si corresponde, asigna un puerto y termina por lanzar el contenedor.

✓ *Implementado completamente.*

Parada del servidor

El usuario puede detener el servidor activo, lo cual implica detener su contenedor sin eliminar los datos del mundo. Esto permite que se conserve el estado del servidor para poder retomarlo más adelante sin ningún tipo de pérdida de información.

✓ *Implementado completamente.*

Inicio de un servidor

Permite volver a iniciar un servidor previamente detenido, o sacarlo del estado de reposo, conservando su estado. Al hacerlo, el sistema reutiliza el contenido existente y reactiva el acceso del mundo.

✓ *Implementado completamente.*

Asignación de puertos

Cada mundo creado recibe automáticamente un puerto disponible. De momento, el cálculo actual se realiza en base al número de directorios existentes, lo que funciona pero podría mejorarse implementando una tabla de control o verificación de puertos en uso.

✓ *Implementado, pero se considera mejorable.*

Gestión de usuarios

Cada usuario tiene su propio espacio de mundos, organizados usando carpetas con el nombre de sus ID interno *(no el nombre del usuario)*. Esto evita conflictos de nombres y mejora un poco la organización. La base de datos PostgreSQL mantiene el registro de los mundos por usuario.

✓ *Implementado completamente.*

Eliminación de mundos

Permite marcar un mundo como eliminado: detiene su contenedor, lo borra y lo renombra internamente para evitar errores. No se elimina físicamente la carpeta de inmediato, sino que se marca como “_DELETED” para una posible recuperación. No se elimina debido a problemas entre medio. También se actualiza la base de datos, la columna “status” como “deleted”.

✓ *Implementado completamente.*

Instalación de mods

Al crear un mundo base, que es las opciones que se da, el usuario puede seleccionar qué mods quiere instalar *(actualmente esto está en una página aparte de la creación normal por plantillas).* Estos se copian automáticamente desde un directorio de mods disponibles a la carpeta del mundo. Esto permite que se haga una personalización sencilla y rápida, similar a los servicios como Aternos.

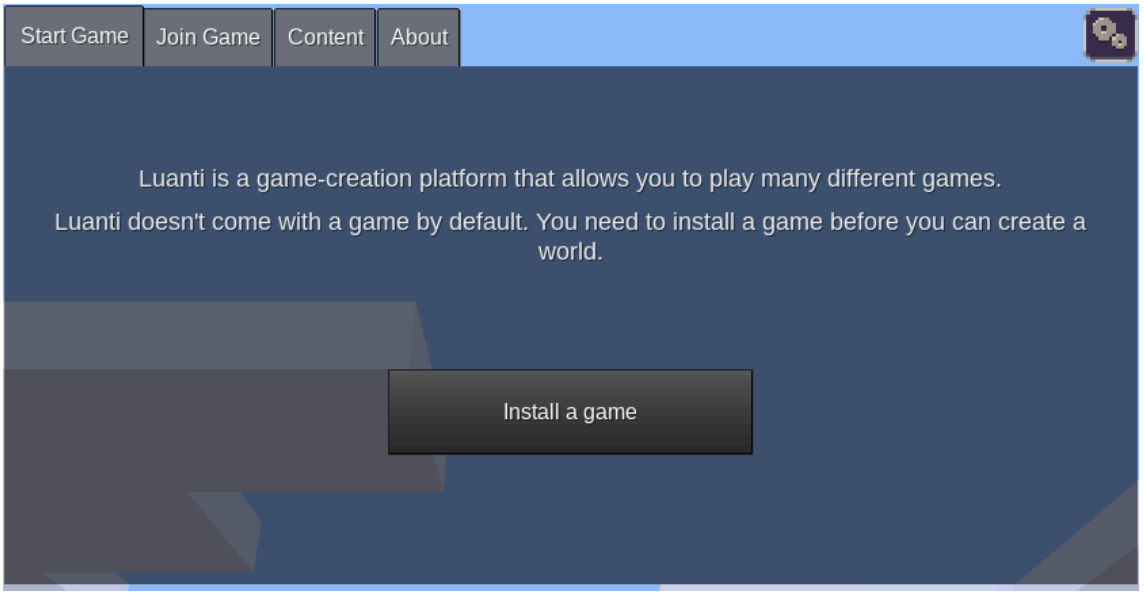
✓ *Implementado completamente.*

Todas estas funcionalidades están diseñadas para ejecutarse desde una interfaz web amigable, sin necesidad de que el usuario conozca comandos o detalles técnicos del sistema.

2.3 Preparación para iniciar con Daemon

A este punto, ya se sabe que la idea nació de otra y que la inspiración, al menos en estructura, son páginas como Aternos, que llega a ser la principal inspiración.
Con la idea de momento en mente se comenzó con la investigación superficial y algo más técnica de Luanti. Las demás tecnologías; apache, psql, podman, etc, no fueron estudiadas por el momento a su vez que las pruebas son hechas, al menos al inicio, de manera local en una máquina Windows.

Luanti está enfocado para la creación de contenido por parte de la comunidad, por lo que no puedes abrir el juego y simplemente crear un mundo. En el menú principal te piden que bajes una plantilla para poder crear un nuevo juego. Plantilla que es la que contendrá las modificaciones necesarias para poder recién crear un mundo:



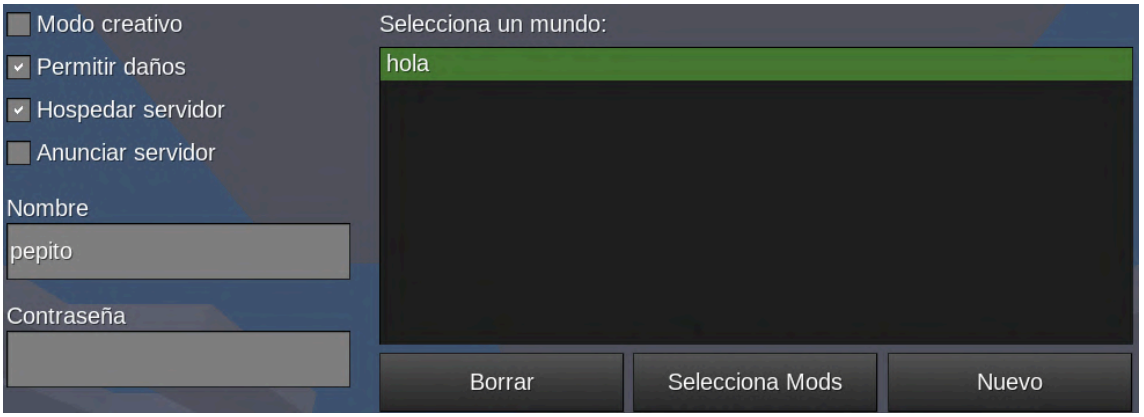
En la pestaña “Content” se nos muestra todo el contenido creado por la comunidad que se categorizan por 3 secciones:

- **Juegos**: Que son las plantillas con la programación *(modificaciones)* necesaria para poder jugar un mundo.

- **Mods**: Que son las diversas modificaciones que se le pueden agregar a los mundos creados, no hay más que decir.
- **Paquetes de texturas**: Que modifican el cómo el juego se va a ver, normalmente las texturas para tener otro aspecto. Esto va por parte del usuario, no del servidor.

Con esto ya aclarado, nos movemos a lo que nos interesa, hostear un servidor.

Luanti te permite crear un mundo con la plantilla que desees y luego hostear el servidor de manera local mediante su cliente:



Y si marcamos “Anunciar servidor” podremos ser vistos desde afuera. Bien esto último trae un par de problemas, como saber si nuestro ISP usa CGNAT lo que nos impide hospedarlo desde casa o que nuestro firewall bloquee conexiones externas. Sin mencionar los ataques que se podrían hacer.

Marcar estos es igual a editar un archivo, el archivo de configuración general de Luanti que se llama “minetest.conf”. **Como aclaración**, varios de los nombres a nivel de código o configuraciones aún se manejan por el nombre “minetest" y no por el reciente Luanti. Es tanto que también se encontraba en el repositorio de los contenedores en GitHub hasta que recientemente lo actualizaron. Continuando, el archivo se ve así:

```
address = 192.168.0.29
name = pepito
remote_port = 31299
maintab_LAST = online
screen_w = 1920
screen_h = 1051
```

Aclarando que es el archivo de un mundo normal creado en el cliente, en un servidor es algo similar. Y lo que añade, lo más importante, es:

- **server_announce = true**: hace que Luanti anuncie tu servidor en la lista de servidores.
- **server_name:** el nombre de tu servidor.
- **server_description:** establece una descripción a tu servidor.

En mi caso no me serviría debido a que usaré contenedores, que se manejan ciertamente similar en cuanto a configuraciones de juego.

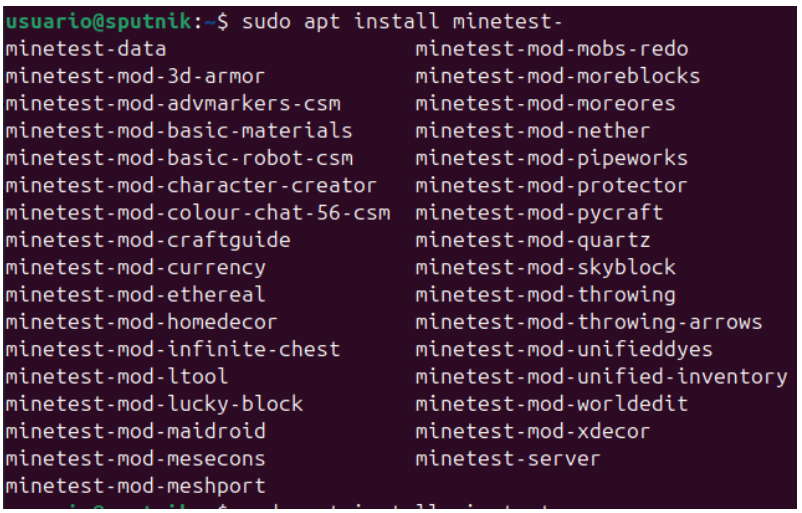
2.4 Contenedores

El entorno en donde se ejecutaría el servidor sería un linux server. Al inicio se estaba pensando en un Desktop, pero se descartó de inmediato ya que el propósito es que el servidor consuma lo menos posible en recursos, con los contenedores como los principales glotones. Se consideró a Debian por un comentario de un foro en donde se decía que esté está más optimizado, y puede que así lo sea, pero al final me decanté por Ubuntu server por la cierta experiencia que tengo sobre él. Se montaría todo en una máquina virtual y hospedaría de manera local.

Las tecnologías hasta ese momento consideradas eran: Podman, para los contenedores en donde correría los mundos; Apache, para alojar la página web que interactuará con el servidor y la base de datos; PostgreSQL, para montar la pequeña base de datos que ayudará a tener un control sobre los usuarios y mundos creados; PHP para el diseño, junto con HTML, de la página web, también este se encargaría de manejar las consultas que se hagan en la paǵina y enviar los datos necesarios para iniciar un mundo.

Digo “hasta el momento” porque, a futuro lo hablaré más a profundidad, consideré utilizar una API hecha en python para que recibiera los datos y este directamente inicie el servidor. Haría uso de Flask y solo actuaría de intermediario, aunque con ello si bien se logró, también considero que se aplastó el enfoque “consumir menos”.

Al inicio, por el archivo que se mencionó anteriormente, se consideró crear un script que simplemente modifique este archivo escribiendo en él lo necesario para hospedar un mundo. Copiando y pegando todo en las carpetas. La idea giraba ya que se pensaba que se podría hacer algo parecido al cliente, y si bien se puede por el comando minetest-server y sus diferentes variantes:



Solo se mantuvo en idea, ya que no veía posible hacerlo, pese a que habían varios tutoriales sobre cómo hospedar un servidor en Ubuntu Desktop.

En su lugar, y de primeras de eso trataba el proyecto, se habló con un profesor que fue el que me informó más del tema de los contenedores. Antes lo habíamos usado, pero no le había visto mucho uso fuera del hecho que se me hizo interesante el poder usarlo con ciertas aplicaciones y que fueran independientes del sistema operativo, pero hasta allí. Cómo llegaban a ser similares a las máquinas virtuales simplemente no los tuve en cuenta.

Luego de que se me explicará mejor, la verdad es que son muy atractivos, explico brevemente:

¿Qué son los contenedores?

Resumiendo que son los contenedores, son como mini-máquinas virtuales, pero más ligeras *(lo que andaba buscando)*. No cargan un sistema operativo completo, sino solo lo necesario para que una aplicación funcione. Usan el kernel del sistema del host pero tienen su propio entorno aislado *(archivo, red, procesos…)*.

En este proyecto, el profesor me enseñó que los de Luanti tenían un contenedor para crear un servidor. Lo había visto muy por encima antes.

Siendo esto así, los usé para levantar servidores Luanti separados para cada mundo con su propio puerto, nada del otro mundo; nombre y configuración. Bueno, así pues, se mantiene más “ordenado”, siendo que el mundo de Pedro no afectará al de Lisa. Cada quién en su cajita feliz y no se molestan.

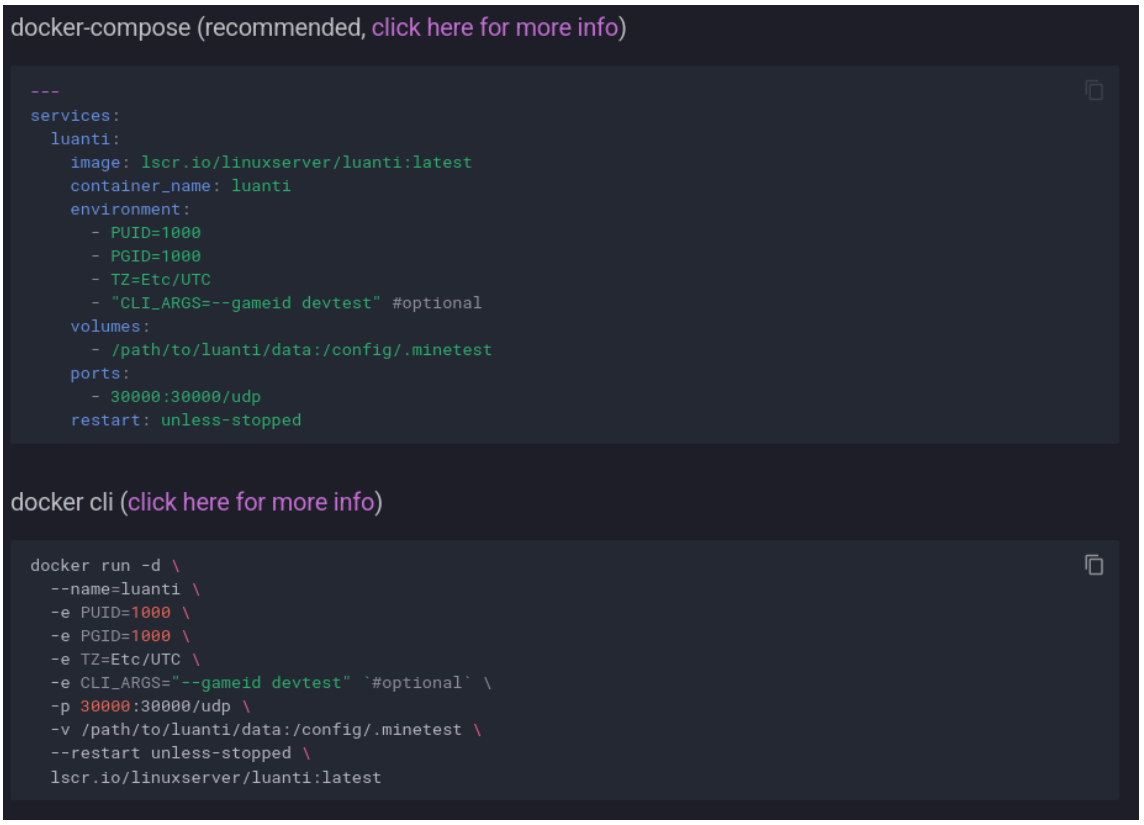
¿Y los Containerfiles?

Un ContainerFile o DockerFile es un archivo o documento de texto que contiene instrucciones para crear una nueva imagen de contenedor.

Si soy sincero, siendo que son una forma más limpia de presentarlo, los tuve en cuenta, sin embargo, fue en parte a que simplemente vi un comando que ya hacía todo lo necesario para desplegar un servidor que no miré ni investigué por completo. Que por cierto, “Containerfile” llega a ser igual a “Dockerfile” así lo anuncian en la página de Red Hat; más a podman al igual que a docker les da igual el nombre del archivo si les especificas cual deben usar.

Sabiendo, más o menos el pilar del proyecto, se empezó con las pruebas.

En la página de linuxserver se nos da una imagen para desplegar un contenedor; en principio usando Docker, pero podman llega a ser equivalente; en el que también nos dicen cómo desplegarlo en un Containerfile y un comando:



Se usó desde el principio el comando. Explicaré el comando que se nos proporciona:

```
podman run -d \
  --name=luanti \
  -e PUID=1000 \
  -e PGID=1000 \
  -e TZ=Etc/UTC \
  -e CLI_ARGS="--gameid devtest" `#optional` \
  -p 30000:30000/udp \
  -v /path/to/luanti/data:/config/.minetest \
  --restart unless-stopped \
  lscr.io/linuxserver/luanti:latest
```

Donde:

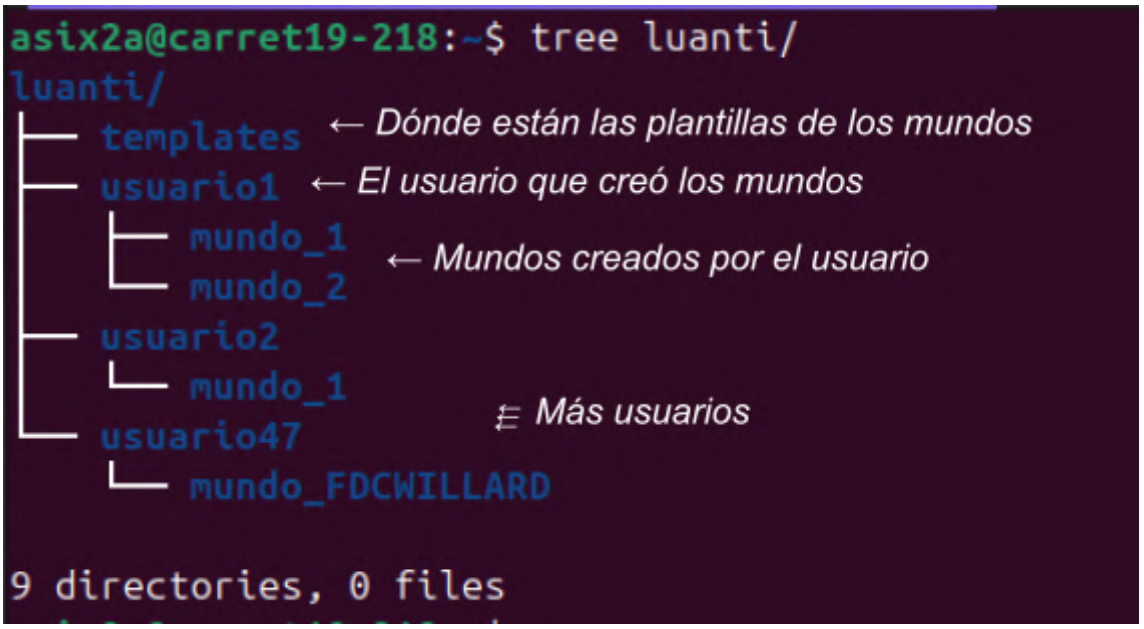
- **podman run -d**: inicia un contenedor en segundo plano.
- **--name=luanti**: le pone nombre al contenedor. Así puedes controlarlo más fácilmente después *(como podman stop luanti)*.
- **-e PUID=1000 y -e PGID=1000**: son los IDs de usuario y grupo que usará el contenedor para acceder a los archivos. En este caso, el usuario normal.
- **-e TZ=Etc/UTC**: define la zona horaria del contenedor.
- **-e CLI_ARGS="--gameid devtest"**: argumentos opcionales para el ejecutable. Aquí se indica usar el juego devtest en lugar del clásico minetest_game. Esta opción es la que se usará para personalizar el mundo en donde se quiera jugar.

- **-p 30000:30000/udp**: Expone el puerto 30000, típico de Minetest, para que se pueda acceder al servidor desde fuera del contenedor.
- **-v /ruta/a/luanti/data:/config/.minetest**: Monta una carpeta local en el contenedor, donde se guardan los mundos, mods, configuraciones... Es decir, el contenido único de cada mundo.
- **--restart=unless-stopped**: El contenedor se reinicia automáticamente si se apaga *(excepto si lo detienes manualmente)*.
- **lscr.io/linuxserver/luanti:latest**: La imagen que se usará, descargada desde el registro de LinuxServer.

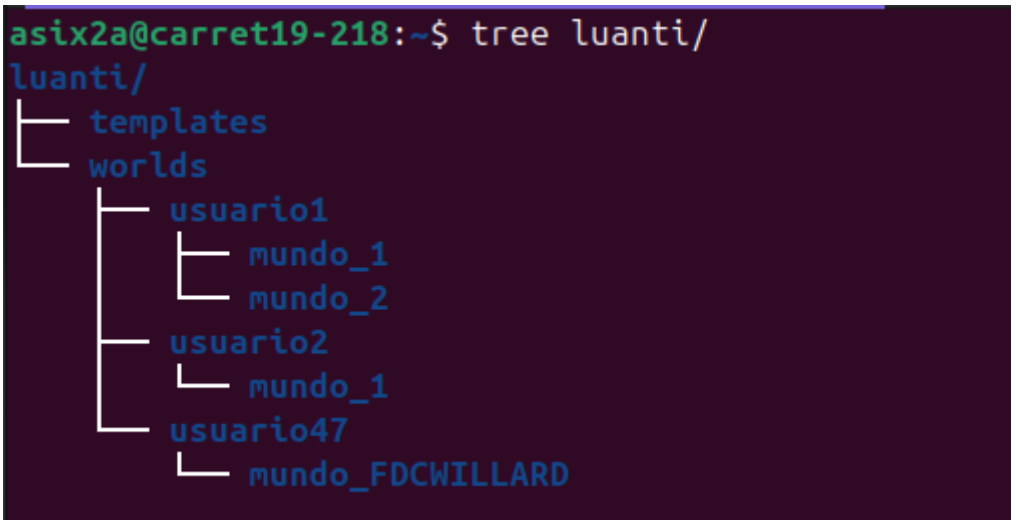
Siendo que ya tengo lo necesario en un solo comando, simplemente me decante por él. No es lo mejor lo tengo que reconocer, solamente que fui muy terco en que se podría usar esto para la automatización, que con un Containerfile también se podría.

Habiendo cubierto lo básico, tocaba hacer un Bash script que se encargará de automatizar todo.

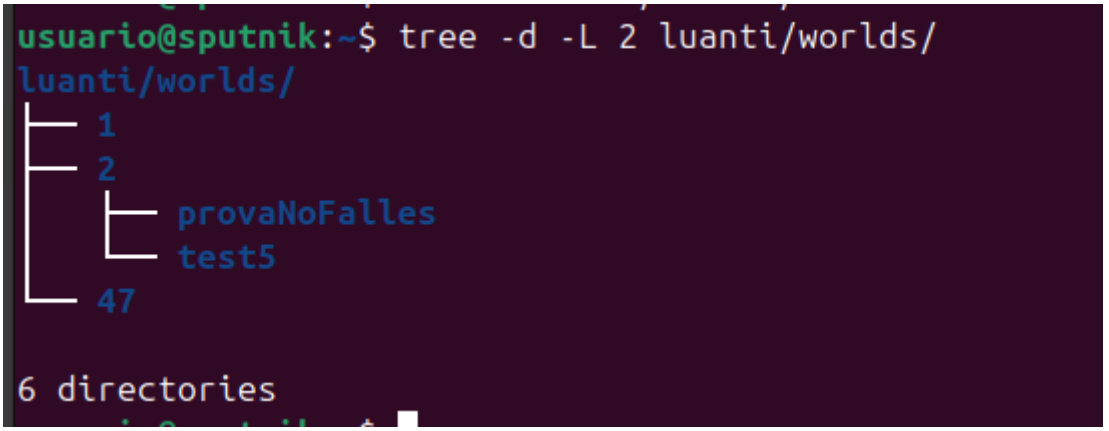
Como los mundos se iban a montar en una carpeta específica se hizo una estructura de carpetas para mantener cada mundo en su propia carpeta de manera ordenada, la primera versión fue de esta manera:



No está mal, Luanti y con el script bien hecho se encargarían de encontrar el mundo correcto. Pero no es cómodo a la vista y aparte habría múltiples usuarios. ¿Qué pasaría si dos usuarios tienen nombres similares? Antes de, lo meto en una carpeta “worlds”:



No se arregla el inconveniente, solo se hizo para más comodidad, para arreglarlo se nombra las carpetas de los usuarios con sus IDs:



Cada contenedor se lanzaría montando solo la carpeta del mundo del usuario que lo pide. Mayormente es estético, pero también aísla los mundos entre usuario y permite borrar sin afectar por accidente a otros.

Luego de ello, no cambió la estructura, se mantuvo igual porque funcionó sin problemas. Solo pulí el cómo se guardaba el mundo y evitar desde el script que no permitiera que se repitan los nombres de los mundos *(cosa que en realidad terminé puliendo en la página)*.

La carpeta templates tendría plantillas descargadas de la página de Luanti. Se pensaba hacer que se descargara en el momento, pero se optó por dar varias plantillas predeterminadas para evitar conflictos entre versiones.

Por ejemplo: el mundo VoxeLibre se descarga, se mueve a la carpeta templates y se descomprime allí:


```
asix2a@carret19-218:~/luanti/templates$ ls minetest_game_30744.zip
minetest_game_30744.zip
asix2a@carret19-218:~/luanti/templates$ unzip -q minetest_game_30744.zip
asix2a@carret19-218:~/luanti/templates$ ls minetest_game
game_api.txt  menu                 mods                screenshot.png
game.conf     minetest.conf        README.md           settingtypes.txt
LICENSE.txt   minetest.conf.example schematic_tables.txt utils
```

Que, esa carpeta es la que usamos para poder crear el mundo. Así se testearon múltiples plantillas y se obtuvieron de pruebas siete:

```
usuario@sputnik:~/luanti/templates$ ls
backroomtest    mineclone2  minetest_game  shadow_forest
capturetheflag  mineclonia  nodecore
```

Definido ello, ahora se debería iniciar un contenedor con el comando anterior más la plantilla que se quiere, con el usuario que se quiera y en un puerto libre. El comando sería el siguiente:

```
podman run -d --name="test_world" \
    -e PUID=$(id -u) -e PGID=$(id -g) \
    -e CLI_ARGS="--gameid /home/usuario/luanti/worlds/47/mineclone2" \
    -p 30001:30000/udp \
    -v "/home/usuario/luanti/worlds/47:/config/.minetest/games" \
    lscr.io/linuxserver/luanti:latest
```

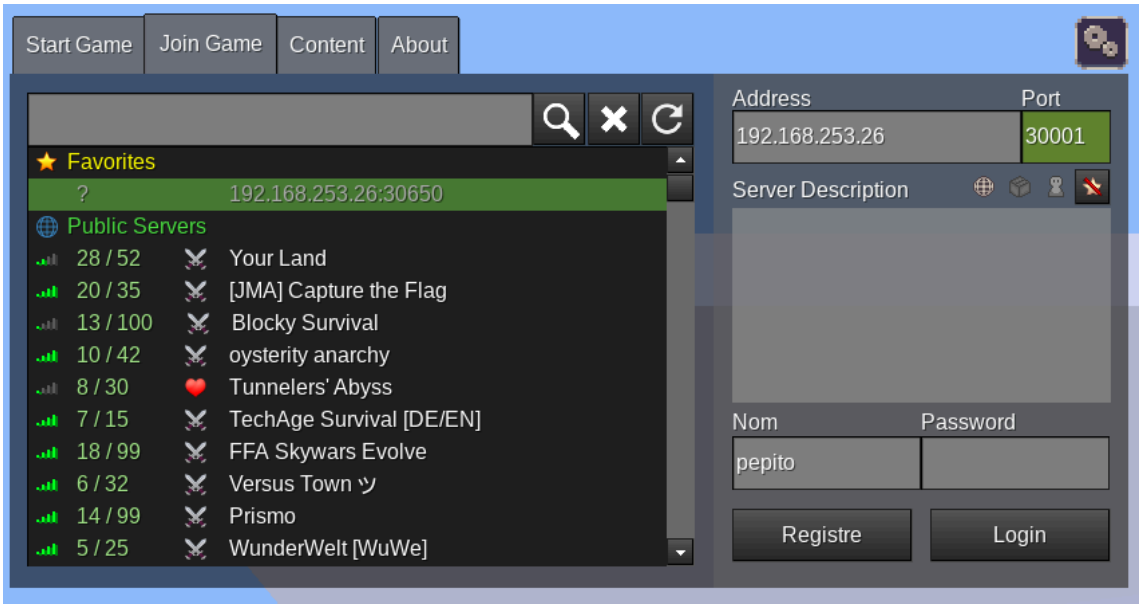
No se agrega nada nuevo, solo se reemplaza con lo que se necesita. Dejo afuera dos parámetros, que son restart y TZ, porque no los consideré necesarios. Eso sí, antes de montar el mundo, se tiene que copiar la carpeta minetest *(que puede ser una plantilla*

personalizada o una de la comunidad, en el ejemplo se usa mineclone2) dentro de la carpeta del usuario para que se pueda usar:

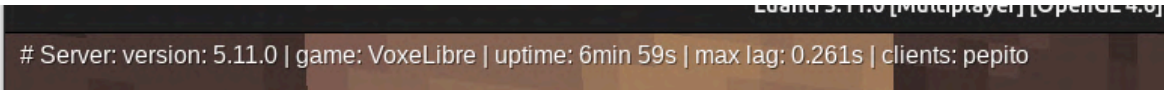
```
usuario@sputnik:~$ podman run -d --name="test_world" \
    -e PUID=$(id -u) -e PGID=$(id -g) \
    -e CLI_ARGS="--gameid /home/usuario/luanti/worlds/47/mineclone2" \
    -p 30001:30000/udp \
    -v "/home/usuario/luanti/worlds/47:/config/.minetest/games" \
    lscr.io/linuxserver/luanti:latest
313d7112a49eaf0fb6c071736188f55089eace4f1eb8f019cce1e2ce1cc19ff8
```

```
usuario@sputnik:~$ podman ps
CONTAINER ID  IMAGE                                COMMAND     CREATED        STATUS        P
ORTS                     NAMES
313d7112a49e  lscr.io/linuxserver/luanti:latest             6 seconds ago  Up 7 seconds  0
.0.0.0:30001->30000/udp  test_world
```

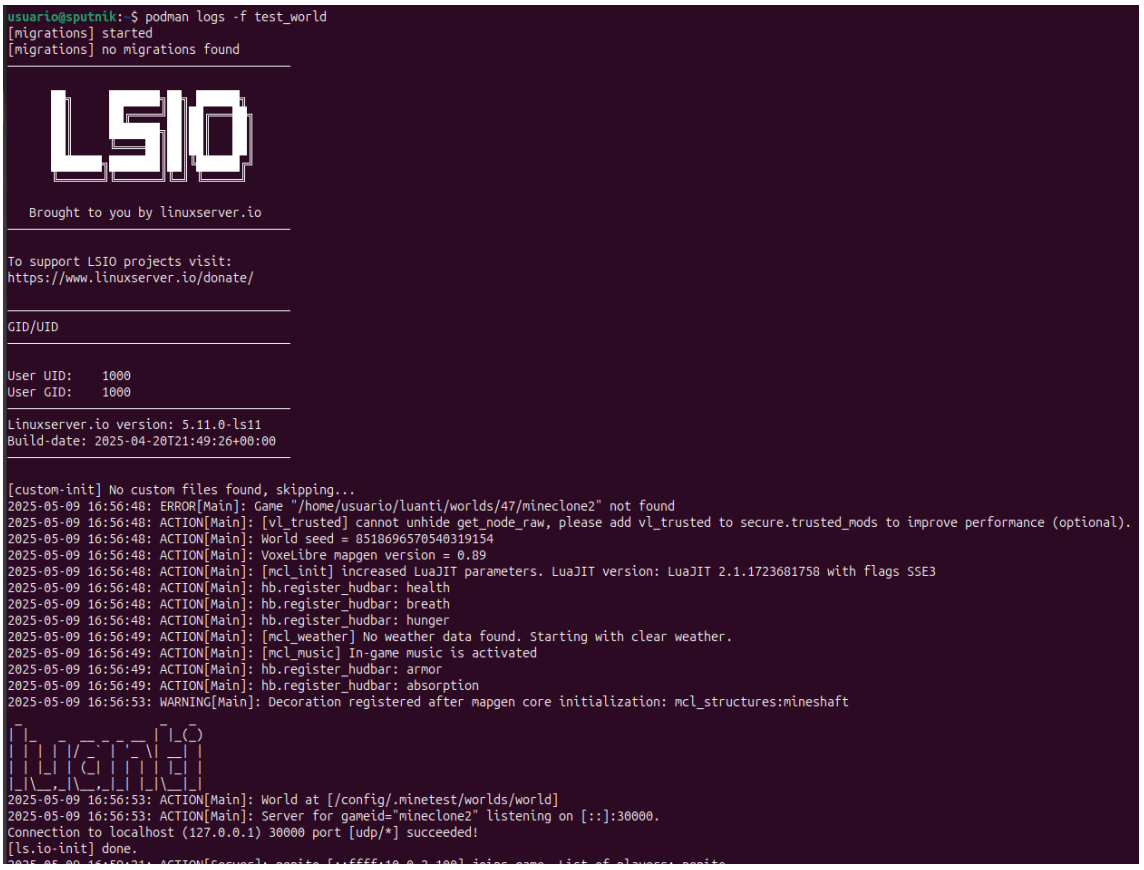
Ahora se podría entrar desde el cliente de Luanti. Como el server se encuentra en una máquina virtual, se pone la IP y el puerto que pusimos en el comando *(30001)*, si no se usa una máquina virtual solo se pondría localhost y el puerto:



Nos registramos y ya estaría jugando en el mundo del contenedor:



También podemos ver los logs del contenedor, aunque esto por comandos en la consola del servidor *(podman logs nombre_del_mundo)*:



Ahora solo sería escribir el script para que lance un contenedor recibiendo ciertos parámetros. La primera versión del script que se escribió fue este *(lo pegaré aquí ya que no es muy grande)*:

```
#!/bin/bash
TEMPLATE_NAME=$1
USER_ID=$2
WORLD_NAME=$3

 # Comprobaciones
if [ -z "$TEMPLATE_NAME" ] || [ -z "$USER_ID" ] || [ -z "$WORLD_NAME" ]; then
    echo "Uso: ./start_server.sh <plantilla> <id_usuario> <nombre_mundo>"
    exit 1
fi

if [ ! -d "./templates/$TEMPLATE_NAME" ]; then
    echo "La plantilla '$TEMPLATE_NAME' no existe en /templates"
    exit 1
fi

USER_PATH="./worlds/$USER_ID"
mkdir -p "$USER_PATH"

WORLD_PATH="$USER_PATH/$WORLD_NAME"
if [ -d "$WORLD_PATH" ]; then
    echo "El mundo '$WORLD_PATH' ya existe para el usuario '$USER_ID'."
    exit 1
fi

# Generación
```

```
mkdir -p "$WORLD_PATH"
cp -r "./templates/$TEMPLATE_NAME" "./$WORLD_PATH/$TEMPLATE_NAME"

BASE_PORT=30000
PORT=$((BASE_PORT + $(find worlds -type d | wc -l)))

CONTAINER_NAME="${USER_ID}_${WORLD_NAME}"

# De momento, si se encuentra un contenedor con el mismo nombre será eliminado
podman rm -f "$CONTAINER_NAME" 2>/dev/null

podman run -d --name="$CONTAINER_NAME" \
        -e PUID=$(id -u) -e PGID=$(id -g) \
        -e CLI_ARGS="--gameid $TEMPLATE_NAME"\
        -p $PORT:30000/udp \
        -v "$(pwd)/$WORLD_PATH/:/config/.minetest/games" \
        lscr.io/linuxserver/luanti:latest
```

Explicándolo, ya que de ahora en adelante solo cambiará algunas cosas:

Parámetros de Entrada

El script recibe 3 parámetros:

- **TEMPLATE_NAME**: Nombre de la plantilla/modo de juego a usar
- **USER_ID**: Identificador del usuario
- **WORLD_NAME**: Nombre del mundo a crear

Comprobaciones Iniciales

- Verifica que se hayan proporcionado los 3 parámetros.
- Comprueba que la plantilla exista en el directorio ./templates/.
- Crea el directorio del usuario si no existe *(./worlds/USER_ID/).*
- Verifica que no exista ya un mundo con ese nombre para ese usuario.

Generación del Mundo

- Crea el directorio para el nuevo mundo *(./worlds/USER_ID/WORLD_NAME/).*
- Copia la plantilla seleccionada al directorio del nuevo mundo.

Configuración del Contenedor

- Calcula un puerto basado en el número de mundos existentes *(a partir de 30000).*
- Define un nombre único para el contenedor combinando USER_ID y WORLD_NAME.
- Elimina cualquier contenedor previo con el mismo nombre *(por si existía).*
- Inicia un nuevo contenedor Podman con lo ya explicado.

Flujo de Trabajo

1. Un usuario solicita crear un nuevo mundo
2. El sistema verifica los parámetros y permisos
3. Se prepara el entorno copiando una plantilla base
4. Se inicia un contenedor aislado para ese mundo específico
5. El contenedor queda ejecutándose en segundo plano con su puerto único

Se creó así para probarlo de manera local. Funciona, aunque de nuevo, su diseño cambiaría manteniendo lo básico.

Ahora, uno de los problemas que surgieron fue un tema con el usuario que se le asignaba a una carpeta.

Espero haber entendido bien el problema. Al momento de querer mover o eliminar los archivos de una de las carpetas montadas en worlds, no se iba a poder porque se les asignaba el usuario “100999” que al parecer se traduciría al usuario “abc”. Aunque al principio no afectaba al montaje, luego y con una actualización del contenedor de Luanti, esto empeoró y no dejaba montar, ni copiar los archivos a la carpeta de destino. En el log de Podman daba errores de permisos:

```
chown: changing ownership of '/config/.minetest/games/mineclone2/textures/mcl_crimson_crims
on_door_top_side.png': Operation not permitted
chown: changing ownership of '/config/.minetest/games/mineclone2/textures/mcl_skins_hair_4.
png': Operation not permitted
chown: changing ownership of '/config/.minetest/games/mineclone2/textures/mcl_mangrove_door
s.png': Operation not permitted
chown: changing ownership of '/config/.minetest/games/mineclone2/textures/mcl_armor_boots_n
```

```
chown: changing ownership of '/config/.minetest/mods': Operation not permitted
[custom-init] No custom files found, skipping...
s6-applyuidgid: fatal: unable to set supplementary group list: Operation not permitted
s6-applyuidgid: fatal: unable to set supplementary group list: Operation not permitted
s6-applyuidgid: fatal: unable to set supplementary group list: Operation not permitted
s6-applyuidgid: fatal: unable to set supplementary group list: Operation not permitted
```

Por lo que pude ver y con ayuda de un profesor se trata sobre cómo Podman maneja los usuarios en contenedores rootless, especialmente cuando se usan volúmenes montados desde el host.

Cuando se ejecuta Podman sin ser root, se utilizan namepaces de usuarios para aislar los contenedores. Los UIDs/GIDs dentro del contenedor se mapean a un rango de sub-UIDs/GIDs *(esto no lo termino de entender bien)* en el host. Y por defecto, Podman asigna un usuario del contenedor con un UID alto en el host.

Como el contenedor es para Docker y este se ejecuta como root, no hace este mapeo de usuario, por lo que si el usuario es el 1000, todos los archivos aparecerán con el UID 1000 en el host.

También estaba el hecho de que PUID y PGID son convenciones usadas por algunas imágenes de Docker, pero no son un estándar. Podman no las interpreta automáticamente, lo que hace que dependa de que la imagen las use correctamente.

Se intentó usar el “--userns keep-id” que mapearía mi UID actual dentro del contenedor. Pero no funcionó, es más, leyendo en reddit a los que hicieron lo mismo, tampoco les funcionó.

Y luego está lo que funcionó por un determinado tiempo, hasta que Luanti actualizara su contenedor y por alguna razón, al menos en las máquinas que probé, ya no surtía efecto.

“--user $(id -u):$(id -g)” que fuerza el usuario dentro del contenedor, según Red Hat. Con ello Luanti, después de otra actualización, ya permitía el PUID/PIGD en cuanto a la copia de archivos y al momento de montar la carpeta, aunque el usuario de los archivos dentro de las carpetas montadas sigue siendo 100999:

```
usuario@sputnik:~$ ls -lh luanti/worlds/47/
total 8,0K
drwxrwxr-x 4 100999 100999 4,0K abr 20 23:54 devtest
drwxrwxr-x 9 100999 100999 4,0K may  9 18:56 mineclone2
```

Y al momento que escribo esto, el usuario cambió a otro para los servidores desplegados por el script:

```
usuario@sputnik:~$ ls -lh luanti/worlds/2
total 8,0K
drwxrwxr-x 4 100032 100032 4,0K may  7 16:20 provaNoFalles
drwxrwxr-x 4 100032 100032 4,0K may  8 19:12 test5
```

Pero por las pruebas realizadas, es lo mismo que el anterior usuario. No tengo mucha idea de como ahora el contenedor de Luanti cambia bien los permisos para los montados. Quiero pensar en este cambio que realizaron:

5.11.0-ls10

Repository: linuxserver/docker-luanti · Tag: 5.11.0-ls10 · Commit: 45a700d · Released by: LinuxServer-CI

CI Report:

https://ci-tests.linuxserver.io/linuxserver/luanti/5.11.0-ls10/index.html

LinuxServer Changes:

Initial Release.

Remote Changes:

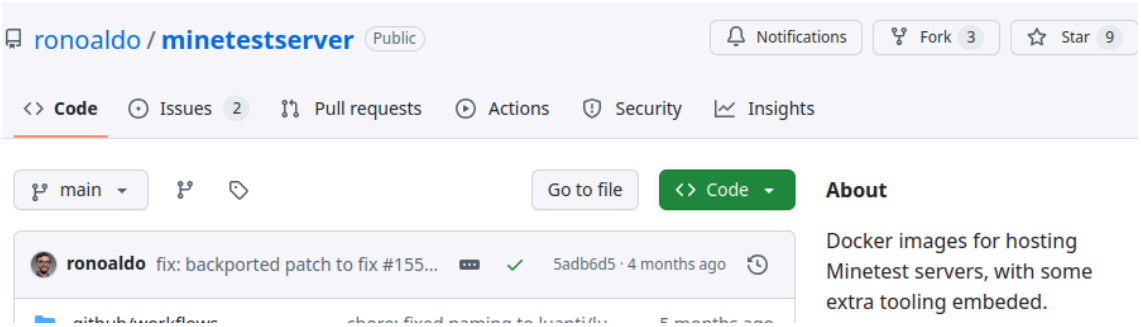
Data change at JSON endpoint https://api.github.com/repos/luanti-org/luanti/releases

Aunque lo dudo mucho. No he encontrado el commit correspondiente más que solo decir que la versión 5.11.0-ls11 funcionó perfecto, y no es que haya habido grandes cambios:

6 package_versions.txt

```
@@ -9,7 +9,7 @@ bash                    5.2.37-r0        apk
 9   9  brotli-libs             1.1.0-r2         apk
10  10  busybox                 1.37.0-r12       apk
11  11  busybox-binsh           1.37.0-r12       apk
12      - c-ares                1.34.3-r0        apk
    12  + c-ares                1.34.5-r0        apk
13  13  ca-certificates         20241121-r1      apk
14  14  ca-certificates-bundle  20241121-r1      apk
15  15  catatonit               0.2.0-r0         apk
@@ -54,8 +54,8 @@ sdl2                    2.30.9-r0        apk
54  54  shadow                  4.16.0-r1        apk
55  55  skalibs-libs            2.14.3.0-r0      apk
56  56  snappy                  1.1.10-r2        apk
57      - sqlite                3.48.0-r0        apk
58      - sqlite-libs           3.48.0-r0        apk
    57  + sqlite                3.48.0-r1        apk
    58  + sqlite-libs           3.48.0-r1        apk
59  59  ssl_client              1.37.0-r12       apk
60  60  tzdata                  2025b-r0         apk
61  61  utmps-libs              0.1.2.3-r2       apk
```

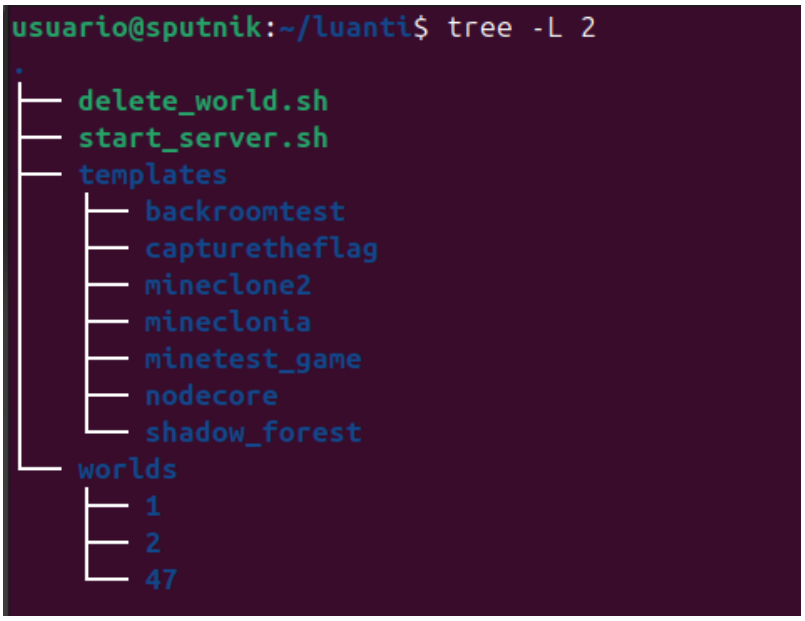
Quizás solo me ahogué en un vaso de agua porque ante el problema, se me presentó otra imagen que el objetivo principal es tomar toda la imagen de Luanti y hacerlo una sola, más personalizable y con herramientas para la búsqueda y descarga de mods. La imagen no ha sido descartada, se sigue teniendo en cuenta:



Intentando ser positivos, ahora mismo con el problema “solucionado” para el montaje de carpetas y con la imagen funcionando se decidió que las carpetas no se eliminarían si el usuario elimina un mundo, los contenedores desplegados sí, pero al aún conservar permisos en la carpeta donde ocurre el montaje, solo se cambiaría a “nombe_DELETED”, para eso se creo un script para eliminar contenedores:

```
#!/bin/bash

USER_ID=$1
WORLD_NAME=$2

if [ -z "$USER_ID" ] || [ -z "$WORLD_NAME" ]; then
    echo "Uso: ./delete_world.sh <ID_usuario> <nombre_mundo>"
    exit 1
fi

CONTAINER_NAME="${USER_ID}_${WORLD_NAME}"

echo "Deteniendo contenedor..."
podman stop "$CONTAINER_NAME"

echo "Eliminando contenedor..."
podman rm "$CONTAINER_NAME"

MARKED_WORLD_NAME="${WORLD_NAME}_DELETED"
mv "./worlds/$USER_ID/$WORLD_NAME" "./worlds/$USER_ID/$MARKED_WORLD_NAME"

echo "Mundo '$WORLD_NAME' marcado como eliminado en
'./worlds/$USER_ID/$MARKED_WORLD_NAME'."
```

Finalmente la estructura de tanto archivos como carpeta queda así:



Tanto Apache como Postgres fueron más tranquilos, con los comandos básicos, pero instalando el mod correspondiente para Apache:

```
sudo apt install postgresql postgresql-contrib
sudo apt install apache2 php libapache2-mod-php php8.3-pgsql
```

2.5 Base de datos y usuarios sin correo, pero con identidad

Ahora lo siguiente sería implementar una base de datos sencilla que almacene los usuarios con sus respectivas IDs y demás, mundos con sus respectivos dueños y más datos que se mostrarán. Mencionándolo así suena vago, pero es que en parte lo es. No sería una base de datos enorme, sino una lo suficientemente buena como para almacenar lo suficiente y que de la información necesaria.

Se hizo con inicio de sesión, pero sin correo. Bastaría con el nombre y contraseña. Esto es peligroso, no hace falta decir el porqué, pero para pruebas estaría bien. Así, la tabla “users” se diseñó muy fácil:

id	username	created_at	password
INTEGER NOT NULL	text NOT NULL	TIMESTAMP DEFAULT CURRENT_TIMESTAMP	VARCHAR(255)

Y en base a esto se creó la tabla “worlds”:

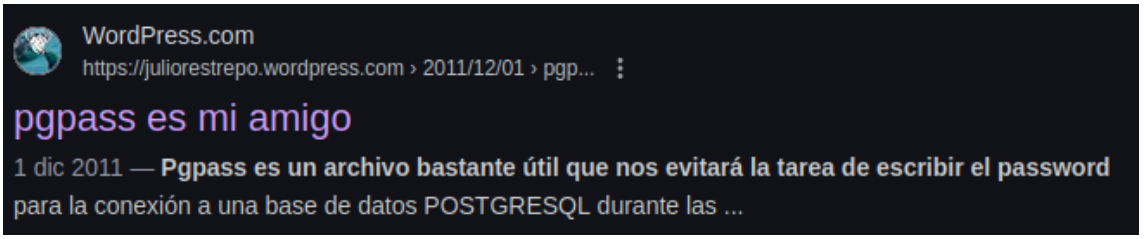
id	user_id	name	template	status	port	created_at
SERIAL PRIMARY KEY	INTEGER REFERENCES users(id) ON DELETE CASCADE	TEXT NOT NULL	TEXT NOT NULL	TEXT DEFAULT ‘active’	INTEGER NOT NULL	TIMESTAMP DEFAULT CURRENT_TIMESTAMP

En donde se intentó registrar lo más importante.

La columna “status” por defecto estaría en “active” porque al momento de iniciar el contenedor este está listo para entrar. Los scripts se encargarían de cambiar este aspecto, por lo que se diseñó 2 más, “start_world” y “stop_world”, iniciar y detener el contenedor respectivamente.

Hasta el momento, todo script, exceptuando “star_server.sh”, solo piden el id del usuario y el nombre que se le ha puesto al mundo.

Se intentó usar el usuario postgres para insertar sin más y sin contraseña usando un archivo llamado “pgpass”:



Pero no dejaban de salir errores:

```
usuario@sputnik:~/luanti$ psql -U postgres -d luanti -c 'SELECT * FROM users'
psql: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed: FAT
AL:  Peer authentication failed for user "postgres"
```

Se intentó cambiar el archivo “pg_hba.conf” de peer a md5, pero no hubo éxito. Al final, se terminó creando un usuario aparte para evitar este tipo de errores. Se le llamó “insert_user” y tenía todo los privilegios sobre la base de datos y sobre los seriales, que siempre daban error.

El archivo “pgpass” es así de sencillo:

```
usuario@sputnik:~/luanti$ cat ~/.pgpass
localhost:5432:luanti:insert_user:usuario
```

Donde cada palabra equivale a: **hostname:puerto:nombredb:usuario:contraseña**. PSQL se encargaría de leer el archivo automáticamente ubicado en el directorio del usuario.

Algo que descubrí, es que en caso se vaya a hacer la página fuera de la máquina virtual, pero en la misma red, se necesita agregar al final de “pg_hba.conf” la línea: **host all all samenet md5.** No es mi caso, pero lo documento.

Teniendo el usuario que insertará y que no da errores, se modificaron los scripts, primeramente “start_server” y “stop_world”. Se resaltarán los cambios más importantes y dejo el script completo en el primer anexo :

Cambios Estructurales Importantes

Rutas Absolutas

- Se definen variables TEMPLATES_DIR y WORLDS_DIR con rutas absolutas, lo que elimina la dependencia de la ubicación de ejecución del script.

Limpieza del Nombre del Mundo

- **Nueva línea**: WORLD_NAME=$(echo "$WORLD_NAME" | tr ' ' '_'). Que reemplaza espacios por guiones bajos para evitar problemas en rutas.

Registro en PostgreSQL

Se añade integración con la base de datos PostgreSQL para registrar:
- ID de usuario.
- Nombre del mundo.
- Plantilla usada.
- Estado del mundo.
- Puerto asignado.

Manejo de Errores Mejorado (más o menos)

Estructura condicional para verificar éxito de:
- Inicio del contenedor Podman.
- Inserción en la base de datos.

Cambios en la Configuración del Contenedor

Usuario/Grupo Fijo

- Cambio de $(id -u)/$(id -g) a valores fijos (1000)

Registro en la base de datos

Sistema de Registro

```
psql -h localhost -U insert_user -d luanti -c \
      "INSERT INTO worlds (user_id, name, template, status, port) \
      VALUES  ('$USER_ID',  '$WORLD_NAME',  '$TEMPLATE_NAME',  'active',
$PORT);"
```

Y en resumen sería lo más primordial. Se intenta verificar cada operación crítica y proporcionar un mensaje sobre qué está pasando, o que puede estar pasando. Esto mismo se le hizo a “stop_world” aunque fue más para probar que cambiase exitosamente el estado del contenedor en la base de datos de la columna “status”. Y lo mismo se tiene planeado hacer para los demás scripts.

2.6 Creación de la página web y backend

Que dolor de cabeza es PHP. Luego de ya tener la base de datos y los scripts encargándose de la creación y demás de los contenedores, paso a la creación de la página web.

Mi enfoque fue en que la página web se encargaría también de la creación de los usuarios, o sea registrarse y de la creación de los mundos, inicio, detenimiento y eliminación. Al menos lo básico sería eso.

Sería hosteada por Apache y al entrar en el navegador, por defecto estaría la página. Bien, primero creé archivos vacíos, que fueron “index.php” y “login.php” alojados en el directorio

del usuario dentro de una carpeta llamada “daemon_web”. Y pasé a configurarlo en Apache.

Primero creé un Virtual Host:

```
sudo nano /etc/apache2/sites-available/daemon.conf
```

Y copio lo siguiente:

```
<VirtualHost *:80>
    ServerName daemon.local
    DocumentRoot /home/usuario/daemon_web
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    <Directory /home/usuario/daemon_web>
        Options -Indexes +FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

Lo más importante siendo que el DocumentRoot es “/home/usuario/daemon_web” y que permite entrar desde cualquier dirección.

Habilito el sitio, desactivo el por defecto y reinicio apache:

```
sudo a2ensite daemon.conf
sudo a2dissite 000-default.conf
sudo systemctl restart apache2
```

Luego de eso ya estaría. Bastaría con entrar en “http://direccion_ip” en un navegador. Bien, hay un error que sucede y es que básicamente es un tema de permisos, no le pude tomar captura, pero es un error 403. Se arregla con el comando: **chmod 755 /home/usuario/daemon_web**.

También, durante la investigación vi que podría llegar a suceder otro error que está relacionado con PHP y es sobre que este no recibiera ciertos archivos. Si bien no me ha pasado, documento el cómo solucionarlo:

```
sudo nano /etc/apache2/mods-enabled/php.conf (o la versión de php que se muestre)
sudo nano /etc/apache2/mods-enabled/php8.3.conf
```

Y asegurarse que esté o tenga la línea:

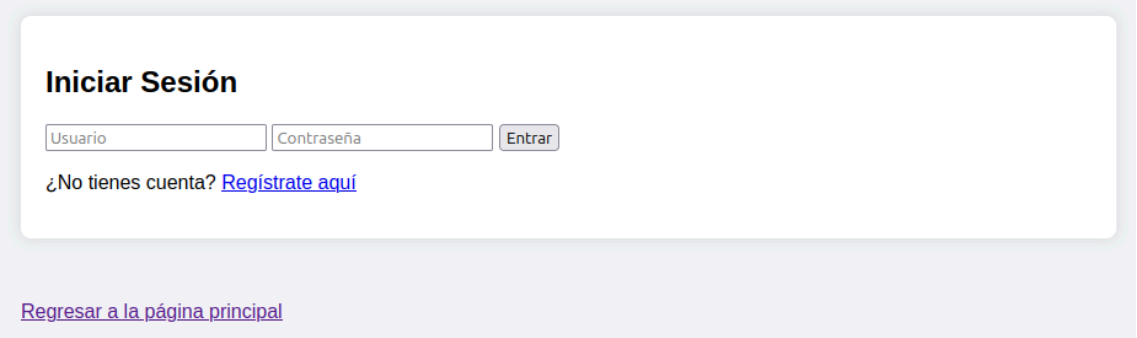
```
<FilesMatch ".+\.ph(?:ar|p|tml)$">
    SetHandler application/x-httpd-php
</FilesMatch>
```

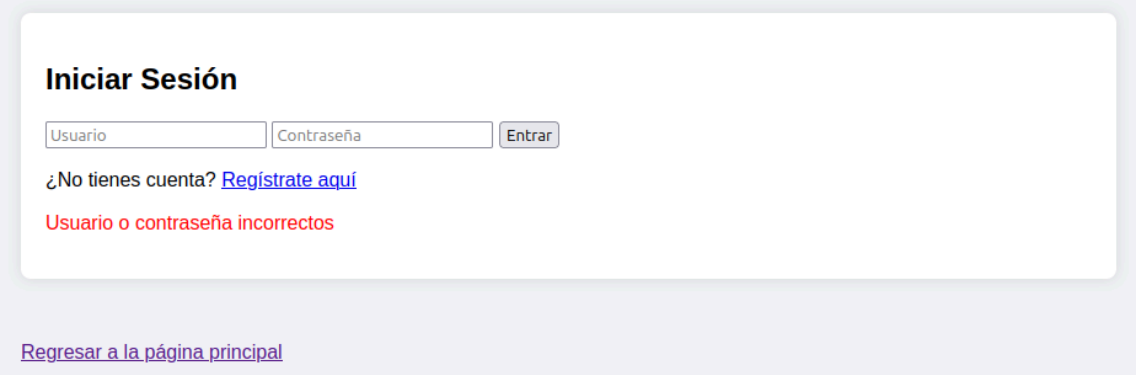
Reiniciar apache y ya estaría. Por el momento, tampoco surgieron problemas con el firewall.

Yendo con los php, creé un nuevo archivo llamado config.php que se encargaría de ser el que se conecte a la base de datos, lo dejo completo en el segundo anexo. Pero resumiendo, hacemos una conexión mediante PHP con la base de datos PSQL llamada luanti. Como la base de datos está en la misma máquina, se pone localhost. Conecto como el usuario “insert_usuario” y la hacemos usando PDO.

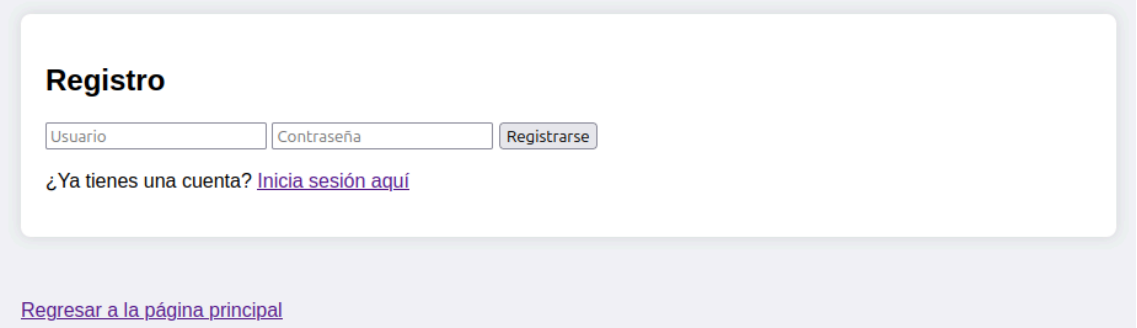
Luego de probar y ver que funcionara fui a por el login. Sí fue una tortura. Para empezar, todas las entradas o páginas web que veía lo hacían con MySQL y aunque era parecido, no era lo mismo. Pero luego de un rato lo conseguí, en el tercer anexo dejo el “login.php”. Tampoco quiero atribuirme todo el mérito, varios de los problemas que se me presentaron no les pude encontrar solución buscando por internet por lo que usé IA para saber en qué me había equivocado o que faltaba.

El login es un php que es almacenado en divs para poder aplicarles estilos más adelante, consulta a la base de datos sobre si hay algún usuario con el username y la contraseña que se introduce, en caso no haber lanza un error avisando que podrían haberse introducido mal los datos:

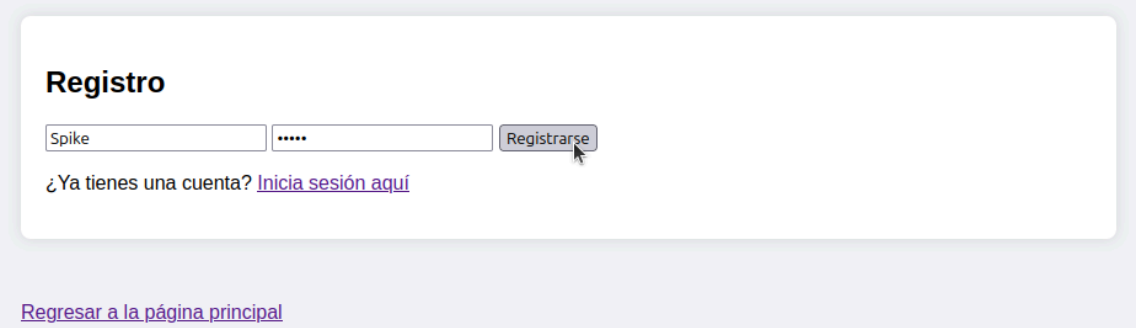


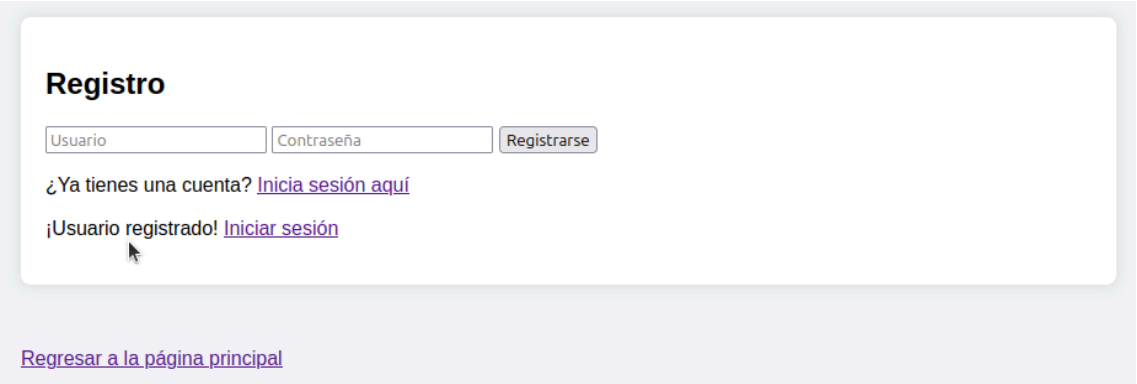


Como se puede apreciar, tiene que haber un usuario creado con anterioridad en el servidor, sino simplemente dará error, para solucionarlo se creó una página para registrarse, dejado en el anexo 4:



Que bueno, el usuario, por ejemplo “Spike” se puede registrar cuantas veces se quiera y con la misma contraseña, pero el ID será diferente en la base de datos. Tema que podría considerar arreglar luego, puesto que de momento no supone ningún problema.





Y en la base de datos se registraría:

```
luanti=# select * from users;
 id | username |         created_at         |                          password
----+----------+----------------------------+------------------------------------------------------------
  2 | lemi     | 2025-05-05 17:20:07.320129 | $2y$10$G1dGSEQuCcA19LWc5EUTheX7i/6H5E6He.HBOs.TtqHTp46dg6Um2
  4 | Spike    | 2025-05-12 20:15:27.097198 | $2y$10$Yue.7/Pd6Py0HybNhkK/dOSIQrJBbhiu9oDQYKMQdxsNWQgiZhMT6
(2 rows)
```

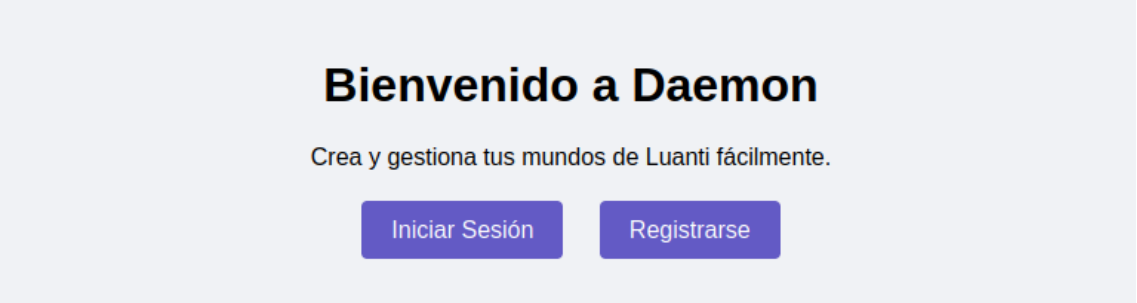
Ah sí, la contraseña se encripta directamente en el envío de datos:

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $_POST['username'];
    $password = password_hash($_POST['password'], PASSWORD_BCRYPT);
```

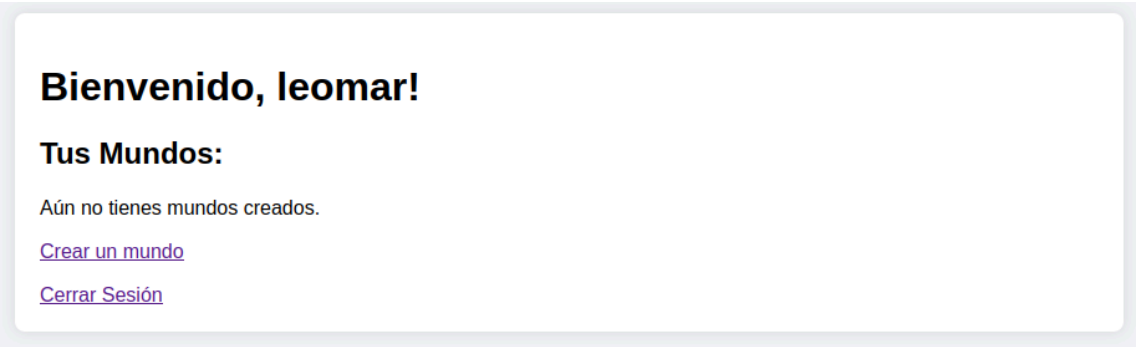
Hago referencias a estas en ambas páginas. Aún no he pulido las webs, y sobre todo, aunque sobre la marcha he ido intentando que no haya muchos problemas de seguridad, uno que varios a de haber. También añadí un “logout.php” muy sencillo.

```
usuario@sputnik:~/daemon_web$ cat logout.php
<?php
session_start();
session_destroy();
header("Location: login.php");
?>
```

Y el index que te da las opciones de iniciar sesión o registrarse:



Luego de iniciar sesión se entra a una página que se llama dashboard:



En el cual muestra información del los mundos que tiene el usuario creados en una lista que se construye al momento:

```
        if (empty($worlds)) {
            echo "<p>Aún no tienes mundos creados.</p>";
        } else {
            echo "<ul class='worlds-list'>";
            foreach ($worlds as $world) {
                echo "<li>
                    <strong>{$world['name']}</strong> |
                    Template: {$world['template']} |
                    Puerto: {$world['port']} |
                    Estado: {$world['status']}
                </li>";
            }
            echo "</ul>";
        }
```

Además del botón de cerrar sesión, está el de crear un mundo y aquí fue donde sí hubo ciertos problemas que no supe resolver o de nuevo, me vi abrumado y busqué una solución apresurada, aquí fue donde como dije al principio siento que se rompió el enfoque “consumir menos”, ya que no conseguí que el usuario “www-data” que es el de apache, pudiera desplegar un servidor.

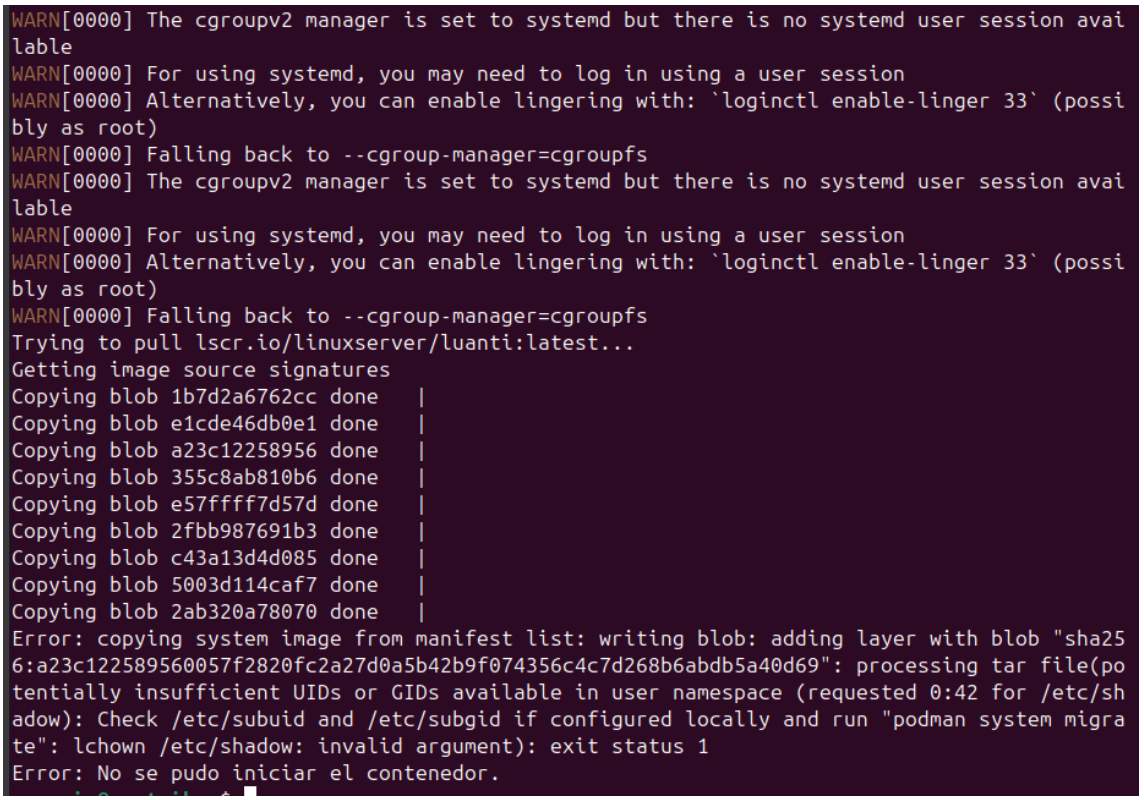
```
usuario@sputnik:~$ sudo -u www-data podman ps
Error: creating runtime static files directory "/var/www/.local/share/containers/storage/li
bpod": mkdir /var/www/.local: permission denied
```

Primero revisé cual eran los problemas:

```
Error: creating runtime static files directory "/var/www/.local/share/containers/storage/li
bpod": mkdir /var/www/.local: permission denied
Error: No se pudo iniciar el contenedor.
```

Todos giraban en torno a que el sistema estaba intentando crear directorios en “/var/www" sin tener los permisos necesarios.

Cómo solución estaba el crear los directorios de manera manual, que eso hice y otorgarle los permisos necesarios para operarlos, si bien el problema se elimina, surge otro:



Aunque sean advertencias, siendo que el primero se puede más o menos descartar ya que no afecta directamente el funcionamiento del contenedor por lo que pude leer. El error está en la segunda parte en donde dice que no hay suficientes UIDs/GIDs disponibles de nombre del usuario. No lo termino de entender directamente.

La solución es modificar “/etc/subuid” y “/etc/subgid” y ampliar las disponibles. Luego usar el comando “podman system migrate” que sirve para realizar cambios en la configuración del sistema. Sin embargo, al menos en mi VM, no funcionó, siendo que se me crasheó y no pude tomar capturas. Luego de intentar prenderlo solo recibe el mismo tipo de error.

Tomé otra ruta a la que llegué por un video que comentaré en el siguiente apartado

2.7 Creación y uso de una API

La ruta que tomé fue la de usar una API que se encargaría de recibir y ejecutar los scripts. La API estaría hecha en Flask que es un framework ligero de python diseñado para crear aplicaciones web y en este caso APIs de manera rápida.

Se encargaría de recibir los datos que serían “plantilla”, “ID del usuario”, “nombre del mundo” y ejecutará el script con ellos. Es por eso mismo que digo que siento que perdió el enfoque de consumir menos, pero al no conseguir lo primero, pensé esta segunda opción como una más viable. De hecho, también se puede crear y gestionar la página entera con Flask, pero decidí que solo usaría para ejecutar los scripts.

Luego de ver como funciona, decidí implementar un endpoint que sería el que controla el inicio de los servidores ejecutando el script. Dejo la primera versión de la API en el anexo.

Explicando por encima lo que hace:

- Defino una ruta con “@app.route(“/start")” y le digo que sólo acepta peticiones POST.
- Extraigo los datos en JSON de la petición.
- Verifico que todos los parámetros estén presentes y retorno un error 400 si falta alguno.
- Preparo el comando con los parámetros pasados, ejecuto el script y capturo la salida.
- Busco indicadores de éxito en la salida y devuelvo 200 para éxito y 500 para errores.
- Manejo los errores mostrandolos directamente en la página.

Usé stdout y stderr solo porque se recomendaba. No sabría con exactitud para qué o cómo funciona, pero por lo que pude ver sirve para el debugging.

Bien quiero comentar que la API captura la salida del script y busca un mensaje de éxito en el, y por eso mismo modifique los scripts para que dieran una respuesta que la API buscaría después de que los datos fueran insertados. No es óptimo, pero me pareció perfecto para lo que llevaba.

Con eso hecho tendría que modificar el cómo se envían los datos en los PHP he invocando a la API:

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
      $template = $_POST['template'];
      $world_name = $_POST['world_name'];
      $user_id = $_SESSION['user_id'];
```

Verificando que la petición sea POST obteniendo los datos y también el ID del usuario de la sesión. Preparo todo tal y como espera la API, configuro headers y el método, y convierto el array con “json_encode”:

```
$data = array(
      'template' => $template,
      'user_id' => $user_id,
      'world_name' => $world_name
);

$options = array(
      'http' => array(
      'header'  => "Content-type: application/json",
      'method'  => 'POST',
      'content' => json_encode($data)
      )
);
```

Luego envío la petición al endpoint Flask y manejo errores con algunos condicionales:

```
$context  = stream_context_create($options);
```

```
$result = file_get_contents('http://127.0.0.1:5000/start', false, $context);
```

Luego de eso, la API ya funcionaba, para iniciarlo solo es de ejecutar:

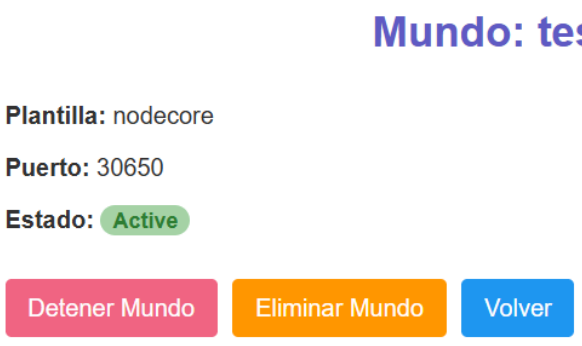
```
usuario@sputnik:~/daemon_web$ python3 api_launcher.py
 * Serving Flask app 'api_launcher'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a prod
uction WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [19/May/2025 15:26:41] "POST /start HTTP/1.1" 200 -
```

Lo convertiré en un demonio o daemon más adelante para que inicie con el sistema.

Por recomendación de un profesor, hice que se pueda entrar a una página aparte del mundo que selecciones y ya desde dentro podrías iniciar, parar o eliminar el mundo. Estos últimos serán realizados también por la API, así que añadí más endpoints.

Creé una nueva página con el nombre “world_detail.php” que se crea al momento de clickear en el mundo que se quiera ver más de cerca. Antes también ordeno los mundos en algo parecido a tarjetas:

Me disculpo por el nombre de los mundos, baja confianza en su funcionamiento. Con esto hecho simplemente modifiqué dashboard y redirección a la página del mundo:



En este punto, también incluí un nuevo script para verificar el estado del mundo, me refiero a si está activo o inactivo. Innecesario, pero lo implementé a manera de prueba para la API, fue más que nada para comprobar si estaba entendiendo bien el cómo funciona. Dejo la API actualizada debajo de la inicial en el anexo.

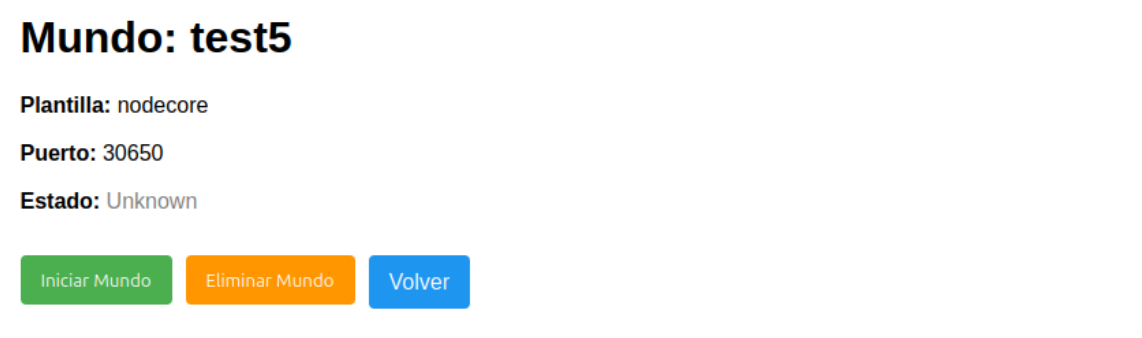
De momento funciona en lo que se necesitaba. No he estado exento de errores, por ejemplo:

Hubo un fallo mío que era básicamente el cambiar de nombre tanto en la carpeta como en la base de datos, esto en sí dentro del servidor no es un problema, pero como tengo este script de prueba que comprueba el estado basándose en el nombre que había consultado en la base de datos, al buscar el mundo por ese nombre cacheado, no lo encontraba y por defecto puse en “world_detail.php” que a se estableciera la palabra “unknown” para mundos no encontrados:

```
127.0.0.1 - - [16/May/2025 15:40:21] "POST /delete HTTP/1.1" 200 -
127.0.0.1 - - [16/May/2025 15:40:21] "POST /status HTTP/1.1" 500 -
```

```
[Fri May 16 15:40:21.926491 2025] [php:warn] [pid 870] [client 192.168.245.218:35452] PHP W
arning:  file_get_contents(http://127.0.0.1:5000/status): Failed to open stream: HTTP reque
st failed! HTTP/1.1 500 INTERNAL SERVER ERROR\r\n in /home/usuario/daemon_web/world_detail.
php on line 39, referer: http://192.168.253.26/world_detail.php?world=borrar2
```

```
// Obtener estado actual
$status_response = call_api('status', $user_id, $world_name);
error_log("Status response: " . $status_response);
$status_data = json_decode($status_response, true);
$current_status = trim($status_data['container_status'] ?? 'unknown');
```



Así que gracias a ese script que hice a modo de prueba supe que solo era modificar en “delete_ world.sh" que cambiara el nombre de la carpeta, pero no del de la base de datos. A este error se sumó otro que era que los json creados incluían saltos de línea:

```
[Fri May 16 16:24:19.481039 2025] [php:notice] [pid 867] [client 192.168.245.218:40922] Sta
tus response: {"container_status":"active\\n","message":"Estado actual: active\\n","status"
:"ok"}\n, referer: http://192.168.253.26/dashboard.php
```

Que se arregla poniendo “trim” en el “$current_status”, esto eliminará espacios y saltos de línea al inicio y final del string. Más resalto, que no los elimina de los registros.

También filtré en el dashboard para que no aparezcan los mundos eliminados *(pondré mundos con la etiqueta DELETED, para que se diferencie más)*:

Modificando la consulta PSQL se arregla:

```php
<?php
$stmt = $conn->prepare("SELECT * FROM worlds WHERE user_id = ? AND status != 'deleted'");
// $stmt = $conn->prepare("SELECT * FROM worlds WHERE user_id = ?");
// SELECT * FROM worlds WHERE user_id = $user_id AND status != 'deleted'
```

También hice que redirigiera al dashboard después de eliminar el mundo:

```php
// Procesar acciones
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $action = $_POST['action'];
    $response = call_api($action, $user_id, $world_name);
    $result = json_decode($response, true);

    if ($result['status'] === 'ok') {
        if ($action === 'delete') {
            header("Location: dashboard.php");
            exit();
        }
        $success = "Acción realizada: " . $result['message'];
    } else {
        $error = "Error: " . $result['message'];
    }
}
```

Pero aún se puede entrar a las páginas de estos mundos escribiendo el nombre del mundo en el buscador:

Se resuelve comprobando que el mundo no tenga la etiqueta “deleted” en la columna “status” de la base de datos:

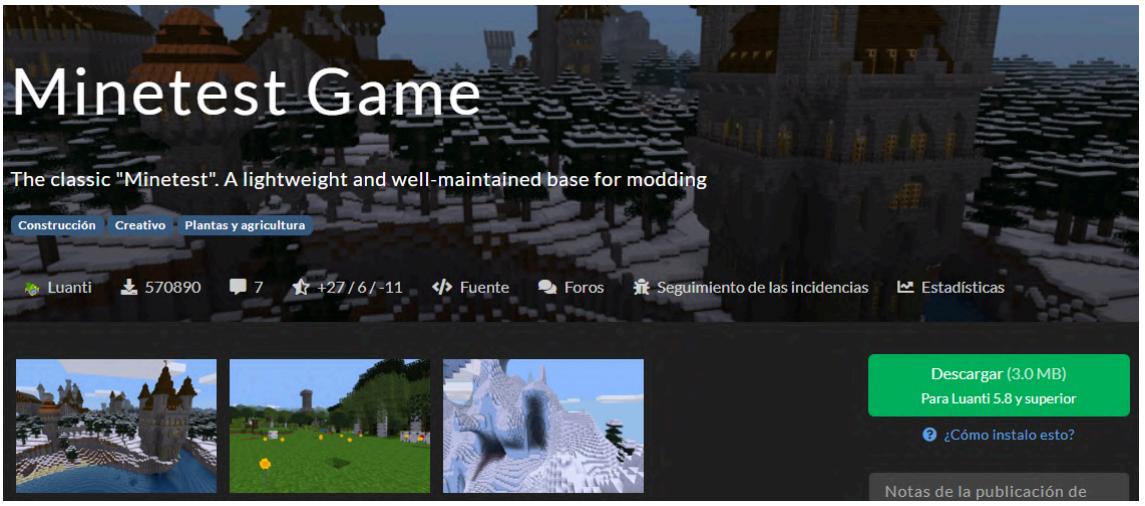
```php
// Esto para evitar que se entre a la página del mundo que ya se había eliminado
if ($world['status'] === 'deleted') {
    header("Location: dashboard.php");
    exit();
}
```

Y lo redirigirá al dashboard nuevamente. Algo que me di cuenta tarde, es que bueno, si eliminas un mundo esté solo te saca de su página y no hay mensaje alguno para hacerlo más amigable.

Con los mundos a base de plantillas ya hechos, era hora de dejar iniciar un mundo con algunos mods.

2.8 Mundos con Mods

Como casi en todo sandbox, Luanti permite a los usuarios crear mods y subirlos en su página oficial. Normalmente el juego te ofrece un “mundo base” o “devtest” que es un mundo el cual posee ciertos elementos por defecto del juego. Ese mundo, es equivalente a una plantilla de un mundo que se llama “Minetest Game” en la página de plantillas:



Dentro de los archivos traen una carpeta especialmente para mods en donde después de descargar el que más te llame la atención, simplemente descomprimes y copias la carpeta resultante dentro de la carpeta mods:

```
usuario@sputnik:~/luanti$ ls worlds/2/modsP1_DELETED/devtest/
game.conf  LICENSE.txt  menu  mods  README.md  screenshot.png  settingtypes.txt
```

Varios mods requieren de otros mods para funcionar, por lo que de momento mostraría mods que no requieran alguna dependencia. Esto es fácil de ver ya que se te indica en la página en un apartado a la derecha. También te recomiendan algunos mods extra y en qué juegos son compatibles, por lo general, todo mod es compatible con el mundo base y las plantilla por defecto también traen mods dentro de ellas que operan, esta información también se te da. A veces es difícil saber si se necesita instalar un mod o ya está por defecto, es más que nada por como cada creador decide dar esta información. Un ejemplo es “player_api” que es un mod que ya viene por defecto el Luanti, algunos te lo remarcan creando confusión, pero nada grave. Como adelanto, no agregaré un sistema que indique

que mods dependen de cual *(porque aunque no ahora, a futuro pondré mods que requieran dependencias, con estas en el selector)* ya que sería tedioso y complicado. Por lo que simplemente el usuario tendrá que visitar la página de Luanti para saber con anterioridad que necesita. Esto de hecho es normal en servicios como Aternos, donde sabes que algo requiere una dependencia o es incompatible porque tu servidor se rompe. Dicho eso, muestro donde se informa de esto normalmente:

Ahora, el plan es usar un script para copiar la plantilla “minetest_game” dentro de una carpeta con el nombre que el usuario escoja, también antes de ello crear una nueva carpeta dentro de la que contiene los scripts *(que en mi caso se llama “luanti”)* llamada “mods” y almacenar mods que no requieran dependencias ya descomprimidos, continuando con el script, vaya a esa carpeta de mods y copie los que se quiera dentro de la carpeta “mods” de la plantilla copiada, y finalmente pasar los parámetros al script de “start_server.sh” para que monte el contenedor en la carpeta modificada, para ello antes tengo que editar una cosa en el script “start_server.sh”, que es:

```
if [ -d "$WORLD_PATH" ]; then
    echo "El mundo '$WORLD_NAME' ya existe. No se sobreescribirá."
else
    mkdir -p "$WORLD_PATH"
    cp -r "$TEMPLATES_DIR/$TEMPLATE_NAME" "$WORLD_PATH/"
    echo "Plantilla copiada al nuevo mundo."
fi
```

Que verifique si la carpeta del mundo ya está creada, en caso lo esté, que no copie la plantilla para que no se eliminen los mods. Después el proceso sigue normalmente.

Ahora pasaría a crear el script para añadir los mods, donde enviaría desde la web *(que no he mencionado, en el siguiente párrafo lo haré)* el nombre de la plantilla que es “minetest_game”, el ID del usuario, el nombre del mundo y un array con los mods seleccionados. Resaltando lo importante del script sería:

```
# Copiamos los mods
mkdir -p "$WORLD_PATH/$TEMPLATE_NAME/mods"
for mod in "${MODS[@]}"; do
    MOD_PATH="/home/usuario/luanti/mods/$mod"
    if [ -d "$MOD_PATH" ]; then
        cp -r "$MOD_PATH" "$WORLD_PATH/$TEMPLATE_NAME/mods/"
    else
        echo "Mod '$mod' no encontrado. Saltando."
    fi
done

/home/usuario/luanti/start_server.sh "$TEMPLATE_NAME" "$USER_ID" "$WORLD_NAME"
```

En donde al final le pasamos a “start_server.sh” el nombre de la plantilla, el ID y el nombre del mundo.

Pasando a la web, al momento de crear un mundo, este nos llevaría a otra página en donde nos hará escoger si crear un mundo en base de una plantilla o crear uno personalizado con la plantilla de Minetest Game:

Luego, internamente sería similar a “create_world.php”, pero con checkboxes y adaptando un nuevo enfoque al momento de mostrar las opciones, siendo que estas no serían añadidas en el html si no que en el php y luego iterando sobre para construirlas:

```
$available_mods = [
    'unified_inventory' => 'Unified Inventory',
    'stamina' => 'Stamina',
    'i3' => 'i3',
    'bonemeal' => 'Bonemeal',
    'worldedit' => 'WorldEdit'
];
```

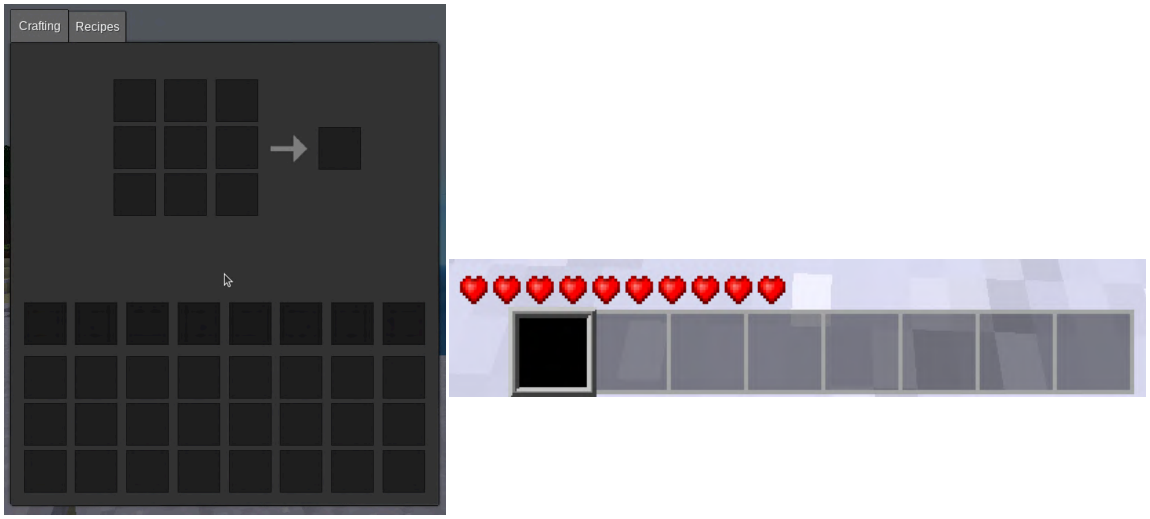
Construimos algo parecido a una grilla con los mods:

```
<div class="form-group">
    <label>Selecciona los Mods:</label>
    <div class="mods-grid">
        <?php foreach ($available_mods as $mod_id => $mod_name): ?>
            <div class="mod-item">
                <input type="checkbox" id="mod_<?php echo $mod_id; ?>" name="mods[]" value="<?php echo $mod_id; ?>">
                <label for="mod_<?php echo $mod_id; ?>"><?php echo $mod_name; ?></label>
            </div>
        <?php endforeach; ?>
    </div>
</div>
```

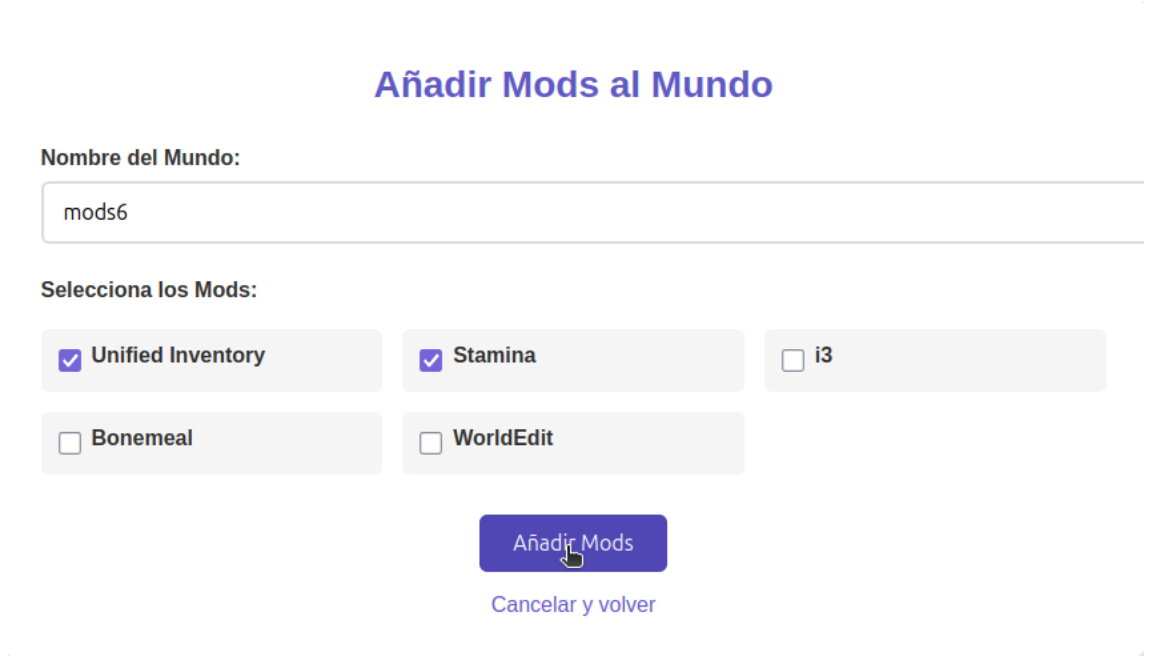
Me aseguraría que al menos se seleccione un mod y preparar los datos en un array, preparar cabecera y demás. En la API creé un nuevo endpoint similar a los anteriores solo que listo para recibir el array. La API leería la salida de “start_server.sh” para saber si se fue bien o hubo un error. Como siempre, la API la dejo en el anexo.

Antes de continuar, me puse a decorar un poco la página sin siquiera saber si funcionaba bien. Pero luego de terminar, sorprendentemente funcionó a la primera. Luego de seleccionar mods que no requieran dependencias, los incluí y realice varias pruebas. Me aseguré de añadir mods que cambian algo visualmente. Para esta prueba escogí dos que cambian el hud del inventario y añaden una barra de stamina en la hotbar:

Luanti vanilla:



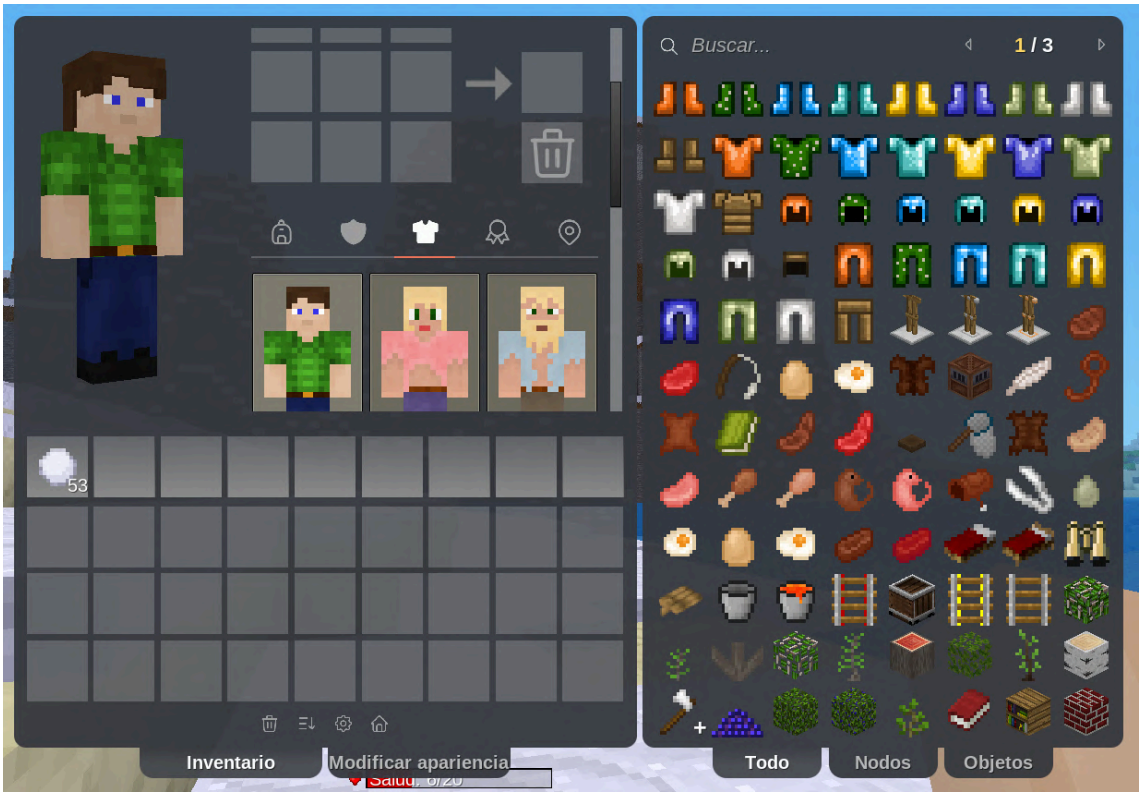
Y desde la página agrego los mods por el formulario:



Luanti con mods:

Luego de esta prueba y ver que funciona bien, me puse a poner más mods he incluso algunos con dependencia y la dependencia requerida, por lo que como ya mencioné, el usuario deberá informarse antes.

No encontré ningún problema o fallo, por lo que dí por hecho esta parte. Agregué más mods y seguí probando. Miré que no hubieran fallos en la selección de mods que necesitan de otros y todo fue bien:



En esta prueba uso cuatro mods: i3, SkinsDB, Awards y Edit Skin. Estos no dependen en sí, pero se complementan entre ellos, así que me queda claro que se está copiando y el contenedor está funcionando bien.

2.9 De API a demonio *(sigue siendo API en realidad)*

Este apartado solo lo usaré para explicar y convertir algo que estaba pasando por alto, y es el de convertir la API en un demonio para que inicie junto con el sistema y no tener que estar haciéndolo manualmente.

Bien, lo convertiremos en un servicio de systemd para que maneje todo en segundo plano, sin necesidad de intervención alguna. La estructura principal es:

1. `[Unit]`: Define las dependencias básicas
2. `[Service]`: Configura cómo ejecutar el servicio

3. `[Install]`: Determina cuándo debe iniciarse el servicio

Y donde haremos que el sistema tome ciertas medidas en cuanto protección y ejecución. Creo un archivo ".service" en el directorio "/etc/systemd/system/" usando sudo:

```
~/daemon_web$ sudo nano /etc/systemd/system/daemon-api.service
```

Y configuro lo siguiente:

```
  GNU nano 7.2                  /etc/systemd/system/daemon-api.service
[Unit]
Description=API Luanti - Lanzador de contenedores
After=network.target postgresql.service

[Service]
User=usuario
WorkingDirectory=/home/usuario/daemon_web
ExecStart=/usr/bin/python3 /home/usuario/daemon_web/api_launcher.py
Restart=always
RestartSec=10

Environment="PATH=/usr/bin"
Environment="PYTHONUNBUFFERED=1"
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

Donde explicandolo sería:

- **Description**: El nombre descriptivo para el servicio, me equivoqué y puse Luanti en vez de Daemon, pero no pasa nada, no influye.
- **After**: Indica que el servicio debe esperar a que:
    - La red esté disponible *(network.target).*
    - El servicio de PostgreSQL esté funcionando.
- **User**: El servicio corre con los permisos de este usuario.
- **WorkingDirectory**: Directorio desde donde se ejecuta el programa, siendo sincero, es muy mala elección de seguridad, pero sirve de momento.
- **ExecStart**: Comando que inicia la aplicación Python.
- **Restart=always**: Si falla, systemd lo reiniciará automáticamente.
- **RestartSec=10**: Espera 10 segundos antes de intentar reiniciar.
- **Environment**: Variables de entorno importantes:
    - **PATH**: Ubicación de los comandos.

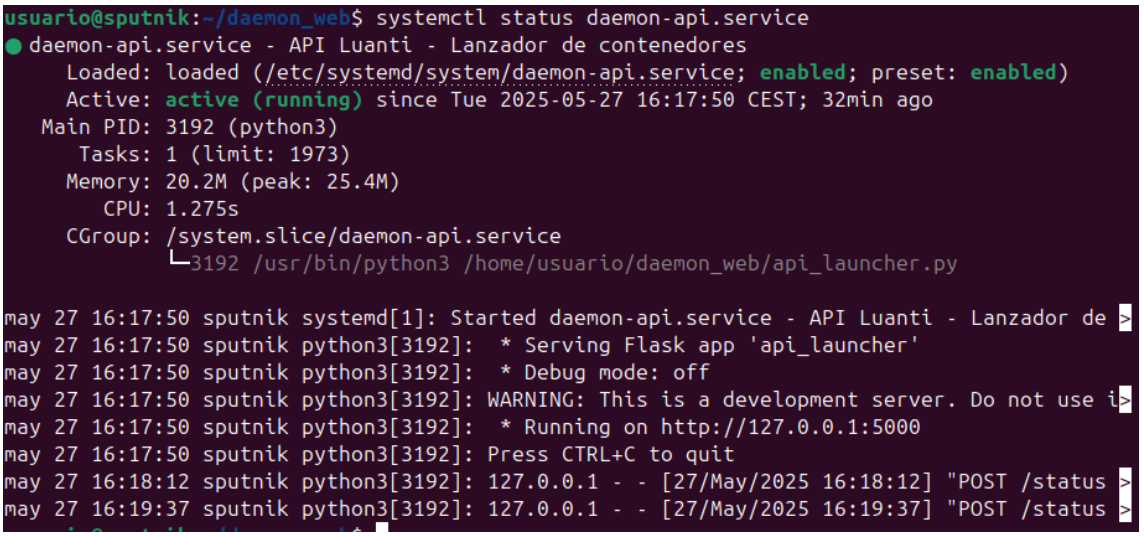
- **PYTHONUNBUFFERED=1**: Evita problemas de buffering en la salida.

- **PrivateTmp=true**: Crea un directorio temporal aislado para mayor seguridad.
- **WantedBy=multi-user.target**: Donde le decimos que debe arrancarse como parte de sistema.

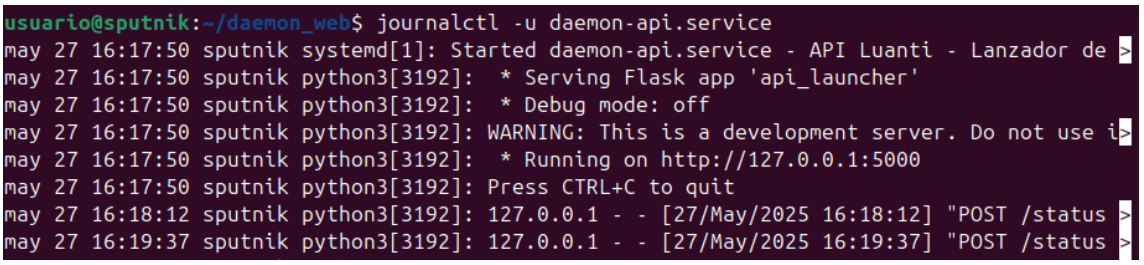
Considero que es lo necesario para el servicio. Para ponerlo en funcionamiento solo sería poner los comandos:

```
sudo systemctl daemon-reexec
sudo systemctl daemon-reload
sudo systemctl enable --now luanti-api.service
```

Y podemos ver su ejecución mediante el comando: **systemctl status daemon-api.service**.



O a mayores rasgos, ver el log completo usando el journal: **journalctl -u daemon-api.service**.



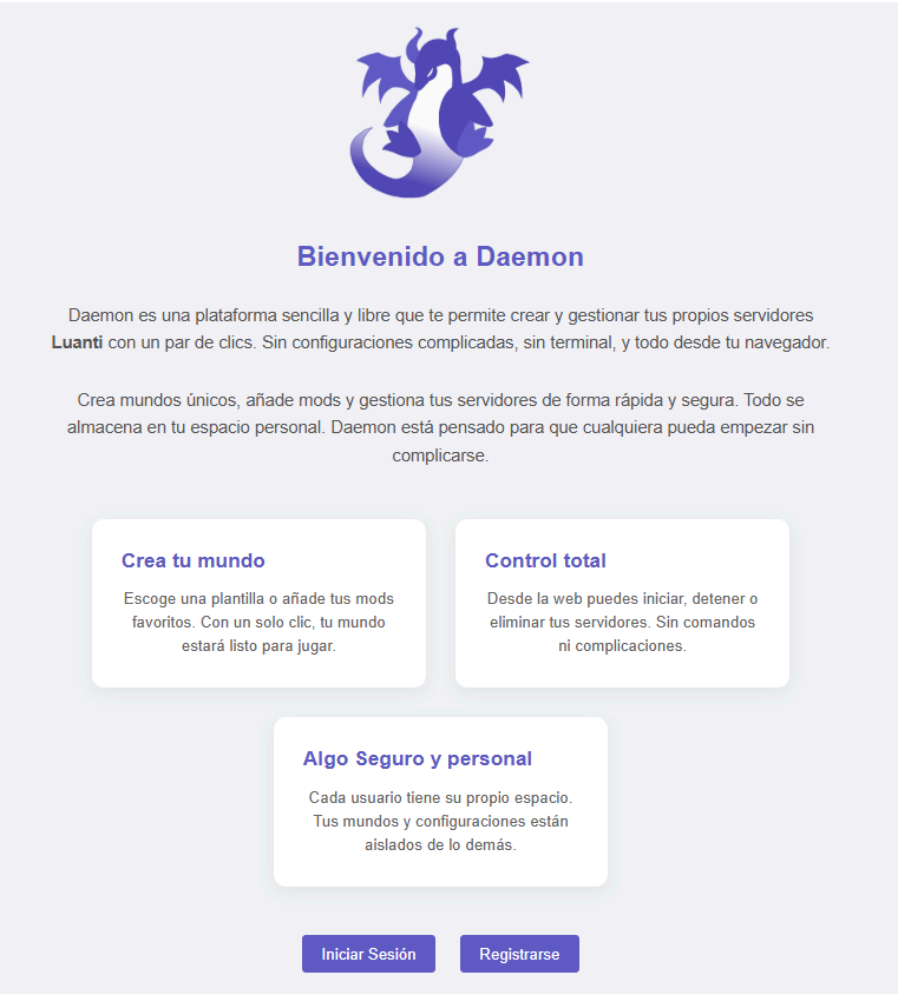
Y poner el parámetro -f para verlo en tiempo real. Se probó y todo funciona perfectamente. Por lo que se le da el visto bueno.

2.10 Estilo de la página web, logotipo, detalles y arreglos posteriores

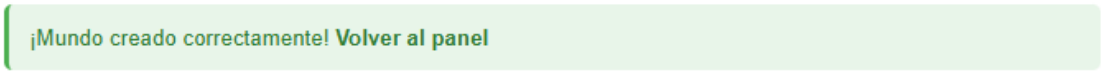
El estilo de la página web ha ido cambiando conforme iba agregando lo esencial, no me quedaba tranquilo con una por lo que me decidía por otro color, estilo o fuentes de letra. Con cada gran adición venían unas horas de relajarse probando estilos para la página,

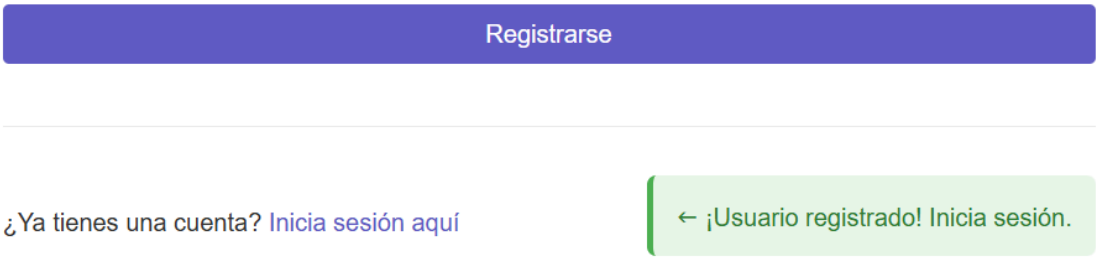
aunque fue frustrante intenté que se reciclara ciertos estilos para diferentes partes. Se pensó en hacer diferentes archivos de estilo, cada uno para su respectiva página y que la principal fuera “style.css” quién tendría una base de algún estilo y mediante otro archivo simplemente cambiaría el color o agregaría cosas nuevas. No lo hice así y decidí poner todo en un CSS he intentar separar por comentarios cada sección. El CSS completo lo dejo en el anexo.

Y en cuanto al logotipo, es una copia de Tux *(el pingüino de Linux)* convertido en un demonio, aunque tiene más parecido a un dragón. El logo iría en “index.php” junto con una explicación de que es Daemon y lo que ofrece:

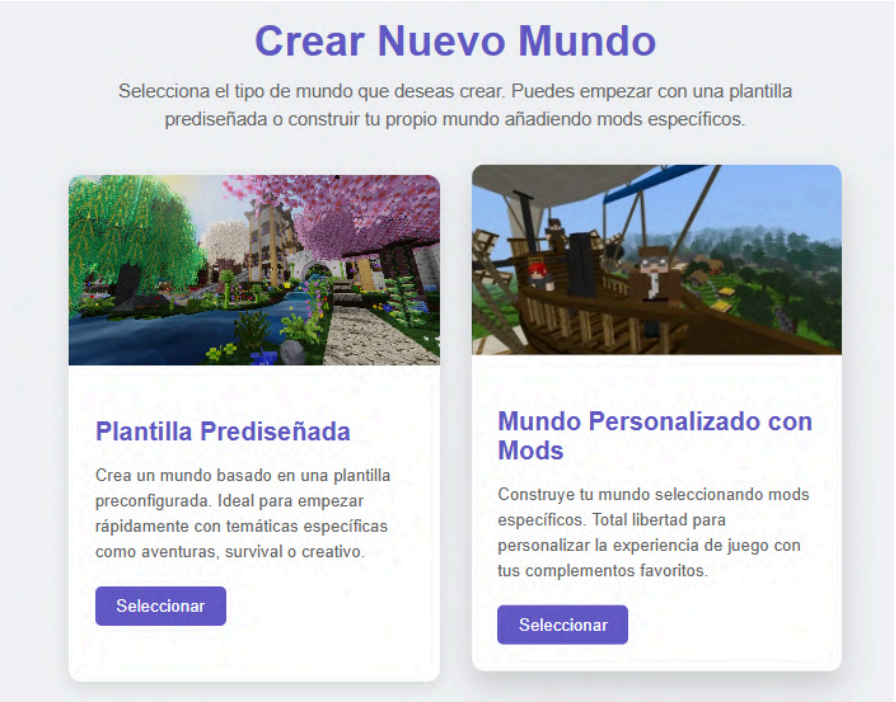


Después me enfoqué en cambiar formas de mostrar errores o mensajes a unos más amigables que no mostraran la salida del script *(aunque todavía se muestra que hizo el script)*:





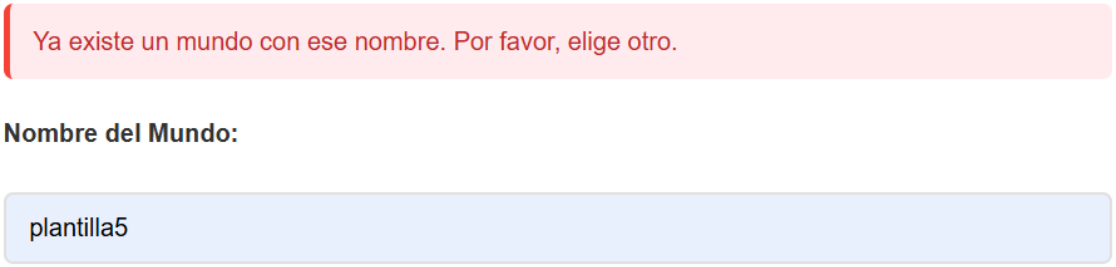
Mejoré la página que nos lleva a decidir qué mundo crear:



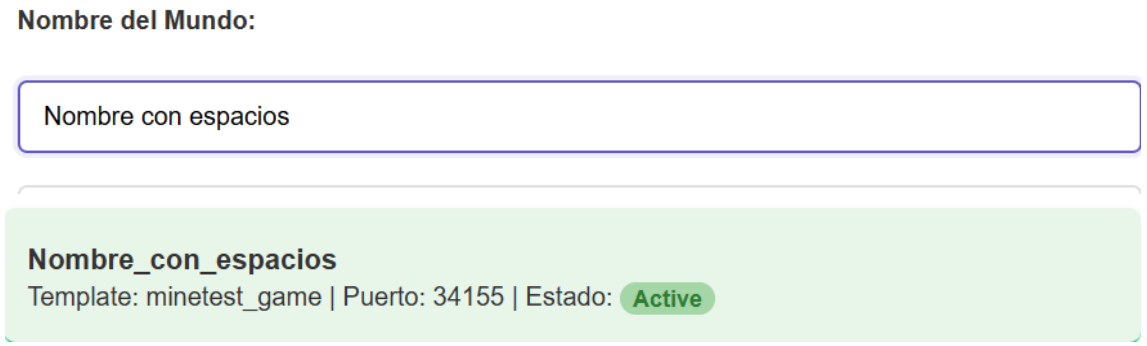
Luego de estar probando un rato que los mundos se crearán bien, decidí que el php evitaría que se insertase un nombre para un mundo dos veces, es decir, llamar a un mundo igual que otro. Por lo que hice que antes de llamar a la API, este primero consultara a la base de datos sobre si hay un mundo asociado al usuario con el mismo nombre:

```php
$stmt = $conn->prepare("SELECT id FROM worlds WHERE user_id = :user_id AND name = :world_name");
$stmt->execute([
    ':user_id' => $user_id,
    ':world_name' => $world_name
]);

if ($stmt->rowCount() > 0) {
    $error = "Ya existe un mundo con ese nombre. Por favor, elige otro.";
} else {
    // Enviar datos a la API
    $data = array(
        'template' => $template,
        'user_id' => $user_id,
        'world_name' => $world_name
    );
```

Esto se hizo para ambos archivos que crean los mundos.



Esto sale solo cuando el usuario crea el mundo, que es donde se hace la verificación. También se comprobó que el script si formateara el nombre si este tenía espacios:



En la base de datos:

Y prácticamente no se hizo ningún otro cambio. En el anexo dejaré un enlace a una carpeta de Google Drive con los scripts y PHPs así como también con las plantillas usadas, mods y el demonio. Se mantendrá en el anexo lo que fue mencionado durante el desarrollo *(por ejemplo en el apartado de la API)*.

4 Conclusions

4.1 Conclusions generals del projecte

El desarrollo del proyecto Daemon ha representado un gran desafío tanto técnico como organizativo. A lo largo de su implementación, no solo consolidé conocimientos ya adquiridos, sino que se abrieron nuevas puertas en ámbitos como la administración de contenedores, automatización mediante scripts *(que aunque tocamos mucho de ello, nunca a este nivel)* y diseño de soluciones web orientadas al usuario final.

Uno de los aprendizajes más significativos diría yo, ha sido la interacción entre sistemas heterogéneos: cómo hacer que una interfaz web en PHP pueda comunicarse de manera eficiente, o lo más que pude, con scripts de bash, contenedores ejecutándose mediante Podman y una base de datos PostgreSQL *(que de este último aprendí y recordé configuraciones y tipos de variables)*. Esto en cierta medida me permitió comprender en profundidad cómo orquestar múltiples tecnologías para formar un sistema funcional, modular y escalable.

También fue clave entender la importancia de la organización estructural del proyecto: establecer jerarquías de carpetas, roles de usuario *(aunque no en gran medida)* y mecanismos de control de errores. Estos aspectos no solo mejoran el mantenimiento del

proyecto, sino que preparan el sistema para una posible ampliación futura o su integración en entornos más complejos.

Yendo al nivel personal y académico, este proyecto ha reforzado mi autonomía y capacidad de aprender nuevas herramientas por necesidad real, ya no solo por teoría. El tener que enfrentarme a errores inesperados, versiones cambiantes de contenedores *(hasta hoy)*, limitaciones del entorno web o problemas de permisos en el sistema operativo, me permitió abordar soluciones con criterio y orden, aunque he de admitir que a veces desesperadas y muy apresuradas, pero siempre tratando de mantener la lógica funcional del proyecto.

Sin dudarlo, Daemon me ha demostrado que la creación de un servicio funcional, desde cero, con múltiples tecnologías, es posible si se fragmenta en objetivos claros, se trabaja con constancia y se mantiene la motivación, incluso en momentos de bloqueo o frustración. La experiencia adquirida será sin duda un pilar para proyectos futuros, tanto dentro como fuera del ámbito. También cabe mencionar que hay diversas maneras de lograrlo, habiendo más o menos eficientes.

4.2 Consecució dels objectius

Yendo al objetivo general que era:

“Diseñar e implementar un sistema automatizado para la creación y gestión de servidores del videojuego Luanti, accesible a los mundos creados por una interfaz web y sustentado por una base de datos relacional, contenedores y scripts personalizados.”

Está ***conseguido***, aunque aún siento que se pueden realizar algunas mejoras, aunque a nivel visual mayormente, la mejora del flujo entre la API y la interfaz web, y la limpieza visual de algunos formularios, así como también se está viendo la seguridad de este. Sin embargo, la arquitectura base ya está establecida y operativa, cumpliendo con los pilares del objetivo general.

En cuanto a objetivos específicos:

- **Automatizar el despliegue de servidores Luanti, estos siendo en su versión contenedor (Podman).**

Conseguido. Se ha logrado automatizar por completo el proceso de creación, inicio y eliminación de servidores, cada uno en su contenedor independiente gracias al uso de Podman y scripts de bash escritos y adaptados a las necesidades del entorno.

- **Permitir que cada usuario pueda crear y gestionar sus propios mundos de forma aislada y segura.**

Conseguido**.** Cada usuario tiene un espacio reservado y aislado tanto en el sistema de archivos como en la base de datos. Los contenedores se nombran con base en el ID del usuario y el nombre del mundo que quiera, y no comparten recursos directos, lo que permite un entorno controlado.

- **Integrar una base de datos PostgreSQL para registrar los mundos y sus respectivos dueños.**

Conseguido. PostgreSQL ha sido correctamente integrado y se utiliza para el registro de información, como los nombres de los mundos, el usuario propietario, el estado *(active, inactive, deleted)* y el puerto asignado, Parte de esta información se consulta y actualiza en la web.

- **Desarrollar una interfaz web en PHP para facilitar el acceso al usuario a la vista de sus mundos, sin necesidad de usar el terminal.**

Conseguido. La interfaz está operativa y permite un inicio de sesión simple, crear mundos, ver sus estados, detenerlos o eliminarlos sin utilizar comandos de consola. También y de improvisto se integró una API interna que ayuda en la comunicación.

- **Establecer, en la medida de lo posible, una estructura de carpetas organizadas por usuarios y mundos, para el almacenamiento local (en el servidor en este caso).**

Conseguido. Se organizan los datos en carpetas con el ID del usuario y el nombre del mundo, permitiendo una gestión clara, y facilitando el mantenimiento o una recuperación de información de ser necesario.

- **Asignar puertos automáticamente a cada servidor (contenedor) desplegado.**

Conseguido, aunque con margen de mejora. Aunque se ha implementado un sistema funcional basado en el conteo de directorios para asignar puertos, se reconoce que este método no es el más robusto. En el futuro se planea sustituirlo por una gestión más precisa, basada directamente en la base de datos o quizás una reserva dinámica, algo parecido.

4.3 Valoració de la metodologia i planificació

Durante todo el desarrollo del proyecto se adoptó una metodología iterativa incremental, lo que me permitió ajustar el enfoque conforme iban surgiendo problemas o ideas nuevas. Si bien al inicio establecí una planificación con tareas semanales y objetivos concretos, en la práctica más de una vez fue necesario modificar algunos plazos, especialmente por factores técnicos no previstos, como los diversos cambios en la imagen oficial de Luanti o en el comportamiento de los contenedores.

A pesar de estos ajustes, la metodología fue más que adecuada, ya que permitió mantener el control del progreso y no perder de vista el objetivo final. Las herramientas de organización como el diagrama de Gantt ayudaron a dividir el proyecto por fases manejables, aunque algunas tareas no se completaron exactamente en las fechas previstas, incluso añadiendo un nuevo trabajo.

Resumiendo, la planificación fue útil como guía general, pero fue necesario aplicar cierta flexibilidad para poder adaptarse a los cambios inesperados. La decisión de trabajar de forma independiente admito que también influyó en el ritmo del desarrollo, lo que exigió una buena gestión de tiempo personal.

4.5 Problemes sorgits i solucions

Dos retos, y lo sigue siendo el primero, fueron entender cómo funcionaba Luanti y los contenedores en Podman, especialmente cuando se trataba de Luanti en contenedor, los parámetros que aceptaba o si la estructura que sigue es similar al la del cliente habitual.

Surgieron problemas de permisos al ejecutar contenedores desde scripts o mediante la interfaz web, lo que me hizo tomar una solución que fue el de usar una API de Python como intermediario, usando Flask, que terminó siendo una gran solución a mi parecer y aprendizaje inesperado. Los problemas de permisos en general sólo constaban de aumentar los permisos para otros.

Otro problema importante fue la modificación de la imagen oficial de Luanti durante el desarrollo. Si bien no he determinado que fue exactamente lo que originó el problema más importante *(el de no poder montar las carpetas)*, tuve una solución temporal que fue la de forzar que Podman determinará el usuario, que me funcionó hasta cierto punto, todo esto hizo que revisara cómo se ejecutaban los contenedores y las diferencias que hay entre imágenes pensadas para Docker y Podman. Junto con ello hubieron problemas de permisos y de puertos. Estos últimos fueron cuando recién comenzaba y estaba reforzando bien como funciona, por lo que no los considero problemas.

Finalmente, en cuanto a diseño y experiencia de usuario, fue algo difícil mantener una coherencia visual y funcional sin contar con herramientas avanzadas en el frontend. Básicamente, que se resumiera todo en un par de clics. Sin embargo, considero que logré un resultado funcional y estéticamente aceptable y amable con el usuario.

4.4 Visió de futur

Aunque el sistema cumple con su funcionalidad básica, existen varias líneas de mejora y ampliación que podrían desarrollarse a futuro. Ejemplos son:

- Mejorar el sistema de asignación de puertos, utilizando algún método que consulte directamente a la base de datos para evitar posibles conflictos.
- Añadir soporte para copias de seguridad automáticas y restauración de mundos. Que no miento al decir que me quedé con ganas de implementar algo parecido a lo de Aternos, linkeado directamente al Drive de la cuenta Gmail que se use.
- Incorporar un sistema de roles o permisos dentro de la plataforma que permita distinguir entre administradores y usuarios normales. Ya que de momento, no hay un administrador más que el que maneje el servidor.
- Ofrecer una sección, o en su defecto mejorar el estilo visual para elegir los mods de forma más intuitiva, que incluya imágenes, descripciones y compatibilidades.
- Integrar un sistema de notificaciones al usuario, algo sencillo, sobre el estado de sus mundos o posibles errores. No solo la barra roja o verde que sale por defecto.
- Desarrollar un sistema de estadísticas y monitoreo del uso de la CPU/RAM de los contenedores en tiempo real. Con ello, permitir que el usuario pueda interactuar con los archivos del juego, que aunque parece romper con lo propuesto, solo sería para gente que quiere más control de su mundo o las configuraciones de los mods.
- Explorar la posibilidad de escalar la solución para entornos externos, utilizando tecnologías como Kubernetes o balanceadores de carga.

- También, otra cosa podría ser la de ofrecer el crear un servidor de algún otro juego. Si seguimos la estética sandbox, Minecraft o Terraria.

Considero que el proyecto ha dejado una base algo sólida sobre la cual se pueden construir muchas más funcionalidades. Además, al estar basado en tecnologías libres, el sistema podría evolucionar como una plataforma comunitaria o educativa.

5. Glossari

Daemon: En el ámbito de la informática, un daemon (a veces traducido como "demonio" en español, aunque no de forma precisa) es un programa que se ejecuta de forma continua en segundo plano, sin la intervención directa del usuario

Contenedor: Entorno virtualizado ligero que encapsula una aplicación y sus dependencias, permitiendo ejecutar múltiples instancias sin interferencias.

PUID/PGID: Identificadores del usuario y grupo en el sistema Linux que permiten controlar los permisos dentro de los contenedores. Utilizado regularmente en Docker.

API (Application Programming Interface): Conjunto de rutas HTTP creadas en Python (con Flask) que permiten comunicar la interfaz web con los scripts del sistema.

Mods: Archivos que amplían o modifican las funciones de un juego. En este caso, se integran en los mundos de Luanti.

Minetest.conf: Archivo de configuración del mundo en Luanti/Minetest que define parámetros como si está activado el fuego, el modo creativo, etc.

Vanilla: En el contexto de los videojuegos, "vanilla" se refiere a la versión base del juego, sin ningún tipo de modificación, actualización, DLC (contenido descargable) ni mods creados por la comunidad.

GUI: En el contexto de los videojuegos, la GUI (Interfaz Gráfica de Usuario) es la parte visual que los jugadores ven y con la que interactúan mientras juegan, como botones, menús, barras de salud, mapas y más.

Stamina: Se refiere a la capacidad de resistencia física o mental *(dependiendo el contexto o videojuego)* para soportar una actividad durante un tiempo prolongado.

6. Bibliografia

No cuento con la fecha en la que he ingresado a los sitios, pero sí con varios de ellos, por lo que los adjunto y también las herramientas empleadas:

Luanti:

Luanti - ContentDB - Help - ContentDB - Luanti Documentation - Server list - Luanti

The Minetest engine internal documentation

luanti - LinuxServer.io - GitHub - linuxserver/docker-luanti

Podman e imágenes:

¿Qué es Podman?

https://hub.docker.com/r/linuxserver/luanti

Understanding rootless Podman's user namespace mode

GitHub - ronoaldo/minetestserver: Docker images for hosting Minetest servers, with some extra tooling embeded.

Error running rootless podman containers inside Incus system container - Ask Ubuntu

Support: podman-compose rootless setup leads to PUID=0 being passed, and ArchiveBox refuses to start as root #1379

La configuración de pgid:puid no funciona, todos los contenedores se ejecutan como root. : r/selfhosted

PostgreSQL

Cómo instalar y utilizar PostgreSQL en Ubuntu 20.04 | DigitalOcean

pgpass es mi amigo

Tipos de datos relevantes en PostgreSQL

PHP

Login básico, paso a paso , en PHP, HTML y PostgreSQL(Validar usuarios con cont…

Validacion inicio de sesion con PHP / Postgresql PDO - Stack Overflow en español

postgresql - Conexion a Postgres des php - Stack Overflow en español

Login y sesiones en un sitio web - Cerrado de la sesión del usuario (logout.php)

Postgresql - PHP - Recorrer Array e insertar en base de datos dependiendo de cada linea del array

PHP: Las funciones definidas por el usuario - Manual

Sesiones. PHP. Bartolomé Sintes Marco - Lección 11: Sesiones | Curso PHP 8.x

PHP include: qué es, cómo funciona y cuándo usarlo | Andrés Ledo

Crear y configurar VHOSTS en Apache - Ejemplos prácticos - ICM

Flask:

Basic Flask Setup With Podman

¿Cómo crear tu propia API con flask? - Blog de Código Facilito

How do I return a subprocess in Flask? - Stack Overflow - Python subprocess module | GeeksforGeeks - flask.jsonify — Flask API

Html, Javascript y CSS:

Uiverse | The Largest Library of Open-Source UI

Browse All the Best Free Tools

PHP: Utilizar un formulario - Manual

HTML DOM Document addEventListener() Method - La regla @keyframes - CSS en español

Herramientas:

ChatGPT

PHP Beautifier and PHP Formatter Online - HTML-Online.com - Editor de HTML, CSS y JavaScript online

FireAlpaca *(para los dibujos)*

Tux - Wikipedia, la enciclopedia libre *(como modelo)*

Aternos

7 Annexos

Script “start_server.sh” con modificaciones para poder insertar en tablas

```
#!/bin/bash

set -x

# Rutas absolutas
TEMPLATES_DIR="/home/usuario/luanti/templates"
WORLDS_DIR="/home/usuario/luanti/worlds"

# Parámetros
TEMPLATE_NAME=$1
USER_ID=$2
WORLD_NAME=$3

# Comprobaciones
if [ -z "$TEMPLATE_NAME" ] || [ -z "$USER_ID" ] || [ -z
"$WORLD_NAME" ]; then
    echo "Uso: $0 <plantilla> <id_usuario> <nombre_mundo>"
    exit 1
fi

if [ ! -d "$TEMPLATES_DIR/$TEMPLATE_NAME" ]; then
    echo "Error: La plantilla '$TEMPLATE_NAME' no existe en
$TEMPLATES_DIR"
    exit 1
fi

# Limpieza del nombre del mundo
WORLD_NAME=$(echo "$WORLD_NAME" | tr ' ' '_')

# Directorio del usuario
USER_PATH="$WORLDS_DIR/$USER_ID"
mkdir -p "$USER_PATH"

WORLD_PATH="$USER_PATH/$WORLD_NAME"

# Crear mundo
# echo Creando mundo "'$WORLD_NAME' (plantilla:
'$TEMPLATE_NAME')..."
```

```
# mkdir -p "$WORLD_PATH"
# cp -r "$TEMPLATES_DIR/$TEMPLATE_NAME" "$WORLD_PATH/"

# Comprobación para saber si el mundo ya se creó, lo que indica
que hay mods
if [ -d "$WORLD_PATH" ]; then
    echo "El mundo '$WORLD_NAME' ya existe. No se
sobreescribirá."
else
    mkdir -p "$WORLD_PATH"
    cp -r "$TEMPLATES_DIR/$TEMPLATE_NAME" "$WORLD_PATH/"
    echo "Plantilla copiada al nuevo mundo."
fi

# Asignar puerto
BASE_PORT=30000
PORT=$((BASE_PORT + $(find "$WORLDS_DIR" -type d | wc -l)))

# Configurar contenedor
CONTAINER_NAME="${USER_ID}_${WORLD_NAME}"
podman rm -f "$CONTAINER_NAME" 2>/dev/null

echo "Iniciando servidor en puerto $PORT..."
if podman run -d \
    --name="$CONTAINER_NAME" \
    -e PUID=1000 \
    -e PGID=1000 \
    -e CLI_ARGS="--gameid $TEMPLATE_NAME" \
    -p $PORT:30000/udp \
    -v "$WORLD_PATH:/config/.minetest/games" \
    lscr.io/linuxserver/luanti:latest; then

    echo "Servidor iniciado para '$WORLD_NAME' (puerto: $PORT).
Registrando en PostgreSQL..."
    psql -h localhost -U insert_user -d luanti -c \
        "INSERT INTO worlds (user_id, name, template, status,
port) \
         VALUES ('$USER_ID', '$WORLD_NAME', '$TEMPLATE_NAME',
'active', $PORT);"

    if [ $? -eq 0 ]; then
        echo "¡Mundo registrado en la BD correctamente!"
```

```
    else
        echo "Error: No se pudo insertar en PostgreSQL. Revisa
conexión o permisos."
    fi
else
    echo "Error: No se pudo iniciar el contenedor."
    exit 1
fi
```

"config.php" para la conexión con la base de datos local

```
<?php
$host = "localhost";
$dbname = "luanti";
$user = "insert_user";
$password = "usuario";

try {
    $conn = new PDO("pgsql:host=$host;dbname=$dbname", $user,
$password);
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    die("Error en la conexión: " . $e->getMessage());
}

?>
```

"login.php"

```
<?php include 'config.php'; ?>
<!DOCTYPE html>
<html lang="es">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Login - Daemon</title>
    <link rel="stylesheet" href="style.css">
    <link rel="shortcut icon" href="/img/logoDaemonHead2.png" />
```

```
</head>

<body>
    <div class="login-box">
        <div class="login-logo">
            <a href="index.php">
                <img src="/img/logoDaemonHead2.png" alt="Logo
Daemon">
            </a>
        </div>
        <h2>Iniciar Sesión</h2>
        <form method="POST">
            <input type="text" name="username"
placeholder="Usuario" required>
            <input type="password" name="password"
placeholder="Contraseña" required>
            <button type="submit" class="btn">Entrar</button>
            <div class="footer-links">
        <p>¿No tienes cuenta? <a href="register.php">Regístrate
aquí</a></p>
        <div>
        </form>

        <?php
        if ($_SERVER["REQUEST_METHOD"] == "POST") {
            $username = $_POST['username'];
            $password = $_POST['password'];

        $stmt = $conn->prepare("SELECT id, username, password
FROM users WHERE username = ?");
        $stmt->execute([$username]);
            $user = $stmt->fetch();

            if ($user && password_verify($password,
$user['password'])) {
                session_start();
                $_SESSION['user_id'] = $user['id'];
                $_SESSION['username'] = $user['username'];
                header("Location: dashboard.php");
        exit();
            } else {
                echo "<p class='alert error'>Usuario o contraseña
```

```
incorrectos</p>";
            }
        }
        ?>
    </div>
</body>
</html>
```

“register.php”

```
<?php include 'config.php'; ?>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Registro - Daemon</title>
    <link rel="stylesheet" href="style.css">
    <link rel="shortcut icon" href="/img/logoDaemonHead2.png" />
</head>
<body>
    <div class="login-box">
        <div class="login-logo">
            <a href="index.php">
                <img src="/img/logoDaemonHead2.png" alt="Logo
Daemon">
            </a>
    </div>
        <h2>Registro</h2>
        <form method="POST">
            <input type="text" name="username"
placeholder="Usuario" required>
            <input type="password" name="password"
placeholder="Contraseña" required>
            <button type="submit"
class="btn">Registrarse</button>
            <div class="footer-links">
                <p>¿Ya tienes una cuenta? <a
href="login.php">Inicia sesión aquí</a></p>
        <div>
        </form>
```

```
        <?php
        if ($_SERVER["REQUEST_METHOD"] == "POST") {
            $username = $_POST['username'];
            $password = password_hash($_POST['password'],
PASSWORD_BCRYPT);

            try {
                $stmt = $conn->prepare("INSERT INTO users
(username, password) VALUES (?, ?)");
                $stmt->execute([$username, $password]);
                echo "<p class='alert success'> ¡Usuario
registrado! Inicia sesión.</p>";
            } catch (PDOException $e) {
                echo "<p class='error'>Error: " .
htmlspecialchars($e->getMessage()) . "</p>";
            }
        }
        ?>
    </div>
</body>
</html>
```

"API"

```
from flask import Flask, request, jsonify
import subprocess

app = Flask(__name__)

@app.route("/start", methods=["POST"])
def start_world():
    data = request.json

    template = data.get("template")
    user_id = data.get("user_id")
    world_name = data.get("world_name")

    if not template or not user_id or not world_name:
        return jsonify({"error": "Faltan parámetros"}), 400

    try:
        command = [
            "/home/usuario/luanti/start_server.sh",
```

```
            str(template),
            str(user_id),
            str(world_name),
        ]
        output = subprocess.check_output(command,
stderr=subprocess.STDOUT).decode()

        if "Servidor iniciado" in output:
            return jsonify({"status": "ok", "message": output}), 200
        else:
            return jsonify({"status": "error", "message": output}), 500

    except subprocess.CalledProcessError as e:
        return jsonify({"status": "error", "message":
e.output.decode()}), 500


if __name__ == "__main__":
    app.run(host="127.0.0.1", port=5000)
```

API actualizada:

```
from flask import Flask, request, jsonify
import subprocess

app = Flask(__name__)


@app.route("/start", methods=["POST"])
def start_world():
    data = request.json

    template = data.get("template")
    user_id = data.get("user_id")
    world_name = data.get("world_name")

    if not template or not user_id or not world_name:
        return jsonify({"error": "Faltan parámetros"}), 400

    try:
        command = [
            "/home/usuario/luanti/start_server.sh",
            str(template),
```

```
            str(user_id),
            str(world_name),
        ]
        output = subprocess.check_output(command,
stderr=subprocess.STDOUT).decode()

        if "Servidor iniciado" in output:
            return jsonify({"status": "ok", "message": output}),
200
        else:
            return jsonify({"status": "error", "message":
output}), 500

    except subprocess.CalledProcessError as e:
        return jsonify({"status": "error", "message":
e.output.decode()}), 500


@app.route("/stop", methods=["POST"])
def stop_world():
    data = request.json

    user_id = data.get("user_id")
    world_name = data.get("world_name")

    if not user_id or not world_name:
        return jsonify({"error": "Faltan parámetros"}), 400

    try:
        command = [
            "/home/usuario/luanti/stop_world.sh",
            str(user_id),
            str(world_name),
        ]
        output = subprocess.check_output(command,
stderr=subprocess.STDOUT).decode()

        if "Contenedor detenido" in output:
            return jsonify({"status": "ok", "message": output}),
200
        else:
            return jsonify({"status": "error", "message":
```

```
output}), 500

    except subprocess.CalledProcessError as e:
        return jsonify({"status": "error", "message":
e.output.decode()}), 500

@app.route("/init", methods=["POST"])
def init_world():
    data = request.json

    user_id = data.get("user_id")
    world_name = data.get("world_name")

    if not user_id or not world_name:
        return jsonify({"error": "Faltan parámetros"}), 400

    try:
        command = [
            "/home/usuario/luanti/start_world.sh",
            str(user_id),
            str(world_name),
        ]
        output = subprocess.check_output(command,
stderr=subprocess.STDOUT).decode()

        if "Contenedor iniciado" in output:
            return jsonify({"status": "ok", "message": output}),
200
        else:
            return jsonify({"status": "error", "message":
output}), 500

    except subprocess.CalledProcessError as e:
        return jsonify({"status": "error", "message":
e.output.decode()}), 500


@app.route("/delete", methods=["POST"])
def delete_world():
    data = request.json

    user_id = data.get("user_id")
```

```
    world_name = data.get("world_name")

    if not user_id or not world_name:
        return jsonify({"error": "Faltan parámetros"}), 400

    try:
        command = [
            "/home/usuario/luanti/delete_world.sh",
            str(user_id),
            str(world_name),
        ]
        output = subprocess.check_output(command,
stderr=subprocess.STDOUT).decode()

        if "Cambio exitoso en base de datos" in output:
            return jsonify({"status": "ok", "message": output}),
200
        else:
            return jsonify({"status": "error", "message":
output}), 500

    except subprocess.CalledProcessError as e:
        return jsonify({"status": "error", "message":
e.output.decode()}), 500


@app.route("/status", methods=["POST"])
def world_status():
    data = request.json

    user_id = data.get("user_id")
    world_name = data.get("world_name")

    if not user_id or not world_name:
        return jsonify({"error": "Faltan parámetros"}), 400

    try:
        command = [
            "/home/usuario/luanti/check_status.sh",
            str(user_id),
            str(world_name),
        ]
```

```
        output = subprocess.check_output(command,
stderr=subprocess.STDOUT).decode()

        if "Error" in output:
            return jsonify({"status": "error", "message":
output}), 404

        return jsonify({
            "status": "ok",
            "container_status": output,
            "message": f"Estado actual: {output}"
        }), 200

    except subprocess.CalledProcessError as e:
        return jsonify({"status": "error", "message":
e.output.decode()}), 500

@app.route("/add_mods", methods=["POST"])
def add_mods():
    data = request.json
    template = data.get("template")
    user_id = data.get("user_id")
    world_name = data.get("world_name")
    mods = data.get("mods", [])

    if not template or not user_id or not world_name:
        return jsonify({"error": "Faltan parámetros"}), 400

    try:
        command = [
            "/home/usuario/luanti/add_mods.sh",
            template,
            str(user_id),
            str(world_name)
        ] + mods

        output = subprocess.check_output(command,
stderr=subprocess.STDOUT).decode()

        if "Servidor iniciado" in output:
            return jsonify({"status": "ok", "message": output}),
200
```

```
        else:
            return jsonify({"status": "error", "message":
output}), 500

    except subprocess.CalledProcessError as e:
        return jsonify({"status": "error", "message":
e.output.decode()}), 500


if __name__ == "__main__":
    app.run(host="127.0.0.1", port=5000)
```

“style.css”

```
/* Estilos base */
body {
    font-family: Arial, sans-serif;
    background: #f0f2f5;
    margin: 0;
    color: #333;
}

/* Contenedores */
.login-box,
.dashboard,
.world-detail {
    background: white;
    margin: 50px auto;
    padding: 30px;
    border-radius: 12px;
    box-shadow: 0 4px 20px rgba(0,0,0,0.08);
    max-width: 800px;
}

.login-box .footer-links {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-top: 20px;
    padding-top: 15px;
    border-top: 1px solid #eee;
}
```

```
.dashboard {
    max-width: 800px;
}

/* Encabezados */
h1 {
    color: #635ac5;
    text-align: center;
    margin-bottom: 30px;
    font-size: 28px;
}

/* Encabezado de sección */
.section-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 20px;
}

.section-header h2 {
    margin: 0;
    font-size: 1.5em;
    color: #635ac5;
}

/* Formularios */
form {
    display: flex;
    flex-direction: column;
    gap: 20px;
}

input[type="text"],
select {
    padding: 12px 15px;
    border: 2px solid #e0e0e0;
    border-radius: 6px;
    font-size: 16px;
    transition: border-color 0.3s;
}
```

```
input[type="text"]:focus,
select:focus {
    border-color: #635ac5;
    outline: none;
    box-shadow: 0 0 0 3px rgba(99, 90, 197, 0.1);
}

input[type="password"],
select {
    padding: 12px 15px;
    border: 2px solid #e0e0e0;
    border-radius: 6px;
    font-size: 16px;
    transition: border-color 0.3s;
}

input[type="password"]:focus,
select:focus {
    border-color: #635ac5;
    outline: none;
    box-shadow: 0 0 0 3px rgba(99, 90, 197, 0.1);
}


/* Lista de mundos */
.worlds-list {
    list-style: none;
    padding: 0;
}

.worlds-list li {
    padding: 10px;
    border-bottom: 2px solid;
}

.nameW {
    font-size: 110%;
    line-height: 28px;
}

/* Items de mundo */
```

```css
.world-item {
   background: white;
   border-radius: 8px;
   box-shadow: 0 2px 4px rgba(0,0,0,0.1);
   margin-bottom: 10px;
   padding: 15px;
}

.world-item[data-status="active"] {
   background-color: #e8f5e9;
   border-bottom-color: #1fd78d;
}

.world-item[data-status="inactive"] {
   background-color: #fff3e0;
   border-bottom-color: #f26483;
}

/* Estados */
.status-active,
.status-inactive,
.status-unknown {
    font-weight: bold;
    padding: 2px 8px;
    border-radius: 10px;
    font-size: 90%;
}

.status-active {
    background-color: #a5d6a7;
    color: #2e7d32;
}

.status-inactive {
    background-color: #ffccbc;
    color: #e65100;
}

.status-unknown {
    background-color: #e0e0e0;
    color: #616161;
}
```

```css
/* Enlaces */
.world-link {
   text-decoration: none;
   color: #333;
   display: block;
   padding: 8px 5px;
}

.world-link:hover {
   background-color: #f5f5f5;
}

.cancel-link {
    display: inline-block;
    text-align: center;
    color: #635ac5;
    text-decoration: none;
    margin-top: 15px;
    transition: color 0.3s;
}

.cancel-link:hover {
    color: #5349b5;
    text-decoration: underline;
}

/* Botones base */
.btn {
    padding: 10px 15px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    text-decoration: none;
    color: white;
    font-size: 16px;
    transition: background 0.3s;
}

/* Botones específicos - aumentamos especificidad */
button.btn.start,
input.btn.start { background: #1fd78d; }
```

```
button.btn.stop,
input.btn.stop { background: #f26483; }

a.btn.create,
input.btn.create { background: #635ac5; }

button.btn.delete,
input.btn.delete { background: #ff9800; }

a.btn.back,
input.btn.back { background: #2196F3; }

/* Hover states para cada tipo */
button.btn.start:hover,
input.btn.start:hover { background: #18b87d; }

button.btn.stop:hover,
input.btn.stop:hover { background: #d9534f; }

a.create:hover,
input.btn.create:hover { background: #5349b5; }

button.btn.delete:hover,
input.btn.delete:hover { background: #e68a00; }

button.btn.back:hover,
input.btn.back:hover { background: #1a7fd4; }

/* Estilo para submit genérico (sin clase específica) */
button[type="submit"],
input[type="submit"]{
    background: #635ac5;
}

button[type="submit"]:not([class]):hover,
input[type="submit"]:not([class]):hover {
    background: #5349b5;
}

/* Alertas */
.alert {
```

```
    padding: 15px;
    margin-bottom: 25px;
    border-radius: 6px;
    font-size: 16px;
}

.success {
    background: #e8f5e9;
    color: #2e7d32;
    border-left: 4px solid #4caf50;
}

.error {
    background: #ffebee;
    color: #c62828;
    border-left: 4px solid #f44336;
}

.success a,
.error a {
    color: inherit;
    font-weight: bold;
    text-decoration: none;
}

.success a:hover,
.error a:hover {
    text-decoration: underline;
}

/* Acciones */
.actions {
    margin-top: 30px;
    display: flex;
    gap: 10px;
    flex-wrap: wrap;
}

/* Home */
.home-container {
    text-align: center;
    max-width: 1000px;
```

```
    margin: 100px auto;
}

.auth-buttons {
    margin-top: 20px;
}

.hero {
    max-width: 800px;
    margin: 0 auto 50px;
}

.hero p {
    font-size: 1.1em;
    line-height: 1.6;
    color: #555;
    margin-bottom: 30px;
}

.features {
    display: flex;
    justify-content: center;
    gap: 30px;
    margin: 50px auto;
    flex-wrap: wrap;
}

.feature-card {
    background: white;
    border-radius: 12px;
    padding: 30px;
    width: 280px;
    box-shadow: 0 5px 15px rgba(0,0,0,0.05);
    transition: all 0.3s ease;
}

.feature-card:hover {
    transform: translateY(-5px);
    box-shadow: 0 10px 25px rgba(0,0,0,0.1);
}

.feature-card h2 {
```

```
    color: #635ac5;
    font-size: 1.3em;
    margin-top: 0;
    margin-bottom: 15px;
    display: flex;
    align-items: center;
    gap: 10px;
}

.feature-card p {
    color: #666;
    line-height: 1.5;
    margin: 0;
}


/* Enlaces del pie */
.footer-links {
    margin-top: 30px;
    display: flex;
    justify-content: space-between;
}

.footer-links a {
    color: #635ac5;
    text-decoration: none;
    transition: color 0.3s;
}

.footer-links a:hover {
    color: #5349b5;
    text-decoration: underline;
}

/*auth buttons*/
.auth-buttons {
    margin-top: 20px;
}

a.btn.login {
    background: #635ac5;
    display: inline-block;
```

```
    padding: 10px 20px;
    margin: 0 10px;
}

a.btn.register {
    background: #635ac5;
    display: inline-block;
    padding: 10px 20px;
    margin: 0 10px;
}

a.btn.login:hover { background: #5349b5; }
a.btn.register:hover { background: #5349b5; }

/* Loader circular pequeño dentro del botón */
.dot {
  animation: blink 1.2s infinite;
  opacity: 0;
}

.dot:nth-child(1) { animation-delay: 0s; }
.dot:nth-child(2) { animation-delay: 0.2s; }
.dot:nth-child(3) { animation-delay: 0.4s; }

@keyframes blink {
  0%, 100% { opacity: 0; }
  50% { opacity: 1; }
}

/* Página para los mods */
label {
    display: block;
    margin-bottom: 8px;
    font-weight: bold;
}

.mods-grid {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
    gap: 15px;
    margin-top: 20px;
}
```

```
.mod-item {
    display: flex;
    align-items: center;
    padding: 10px;
    background: #f9f9f9;
    border-radius: 6px;
    transition: all 0.2s;
}

.mod-item:hover {
    background: #f0f0f0;
    transform: translateY(-2px);
}

.form-group {
    margin-bottom: 25px;
}

#createBtn {
    background: #635ac5;
    color: white;
    border: none;
    padding: 12px 25px;
    border-radius: 6px;
    font-size: 16px;
    cursor: pointer;
    transition: background 0.3s;
    display: block;
    margin: 30px auto 0;
}

#createBtn:hover {
    background: #5349b5;
}

/* Página para la redirección de la creación de mundos */
.creation-options {
    display: flex;
    justify-content: center;
    gap: 30px;
    margin-top: 40px;
```

```
    flex-wrap: wrap;
}

.creation-card {
    width: 350px;
    background: white;
    border-radius: 12px;
    overflow: hidden;
    box-shadow: 0 5px 15px rgba(0,0,0,0.1);
    transition: all 0.3s ease;
    text-decoration: none;
    color: #333;
}
.creation-card:hover {
    transform: translateY(-10px);
    box-shadow: 0 15px 30px rgba(0, 0, 0, 0.15);
}

.card-image {
    height: 180px;
    background-size: cover;
    background-position: center;
}

.template-image {
    background-image:
url('https://blog.luanti.org/static/blog/5.11.0/cover.webp');
}

.mods-image {
    background-image:
url('https://www.kdeblog.com/wp-content/uploads/2024/02/Minetest-
el-juego-juego-libre-basado-en-voxeles-Juegos-Linux-VIII_6.webp')
;
}

.card-content {
    padding: 25px;
}

.card-title {
    font-size: 1.5em;
```

```
    margin-bottom: 15px;
    color: #635ac5;
}

.card-description {
    margin-bottom: 20px;
    line-height: 1.5;
    color: #666;
}

.card-button {
    display: inline-block;
    padding: 10px 20px;
    background: #635ac5;
    color: white;
    border-radius: 6px;
    text-decoration: none;
    transition: background 0.3s;
}

.card-button:hover {
    background: #5349b5;
}

.page-header {
    text-align: center;
    margin-bottom: 20px;
}

.page-header h1 {
    color: #635ac5;
    font-size: 2.5em;
    margin-bottom: 10px;
}

.page-header p {
    color: #666;
    font-size: 1.1em;
    max-width: 700px;
    margin: 0 auto;
    line-height: 1.5;
}
```

```
@media (max-width: 768px) {
    .creation-options {
        flex-direction: column;
        align-items: center;
    }

    .creation-card {
        width: 100%;
        max-width: 350px;
    }
}

/* Estilos para el logo, junto con un hover porque no */
.header-logo {
    display: flex;
    justify-content: center;
    margin-bottom: 20px;
}

.header-logo img {
    height: 200px;
    width: auto;
    transition: transform 0.3s ease;
}

.header-logo img:hover {
    transform: scale(1.05);
}

.login-logo {
    text-align: start;
    margin-bottom: 20px;
}
.login-logo img {
    height: 50px;
    width: auto;
    transition: transform 0.3s ease;
}

.login-logo img:hover {
    transform: scale(1.05);
```

```
}
```

Google Drive de scripts, PHPs, estilo, demonio, imágenes, plantillas y mods utilizados para el proyecto:

Proyecto Daemon