**¿Qué es Laravel?**

* Un **framework MVC** (Modelo–Vista–Controlador) que separa claramente la lógica de negocio (Modelos), la presentación (Vistas) y el flujo de control (Controladores).
* Proporciona una **arquitectura organizada**, con convenciones sobre configuración (“convention over configuration”) para acelerar el desarrollo.

**Características principales**

1. **Artisan CLI**
   * Herramienta de línea de comandos que genera controladores, modelos, migraciones, seeds y más en segundos.
2. **Eloquent ORM**
   * ORM “activo récord” que facilita trabajar con bases de datos: define relaciones, consultas y scopes con sintaxis fluida.
3. **Migraciones y Seeders**
   * Versiona el esquema de la base de datos en código, permitiendo crear, modificar y poblar tablas de forma segura.
4. **Blade Templating**
   * Motor de plantillas ligero y expresivo para las vistas, con herencia de layouts y directivas propias (@if, @foreach, etc.).
5. **Middleware y Rutas**
   * Filtra y agrupa rutas con middleware (autenticación, CORS, rate limiting) de manera sencilla.
6. **Colas y Jobs**
   * Desacopla tareas pesadas (envío de correos, procesamiento de imágenes) para ser ejecutadas en segundo plano.
7. **Eventos y Listeners**
   * Sistema de eventos nativo para desacoplar y extender funcionalidades sin tocar el núcleo de la aplicación.
8. **Seguridad incorporada**
   * CSRF protection, hashing de contraseñas, encriptación, políticas de autorización y validaciones centralizadas.

**¿Por qué usar Laravel?**

* **Productividad**: Artisan y los generadores reducen boilerplate.
* **Mantenibilidad**: Arquitectura limpia y modular, fácil de escalar y refactorizar.
* **Comunidad**: Ecosistema con paquetes oficiales (Sanctum, Passport, Telescope) y una gran comunidad que aporta soluciones.
* **Ecosistema completo**: Herramientas de testing (PHPUnit integrado), Vite/Laravel Mix para assets, scheduler para tareas programadas.

¿Por qué Laravel con React?  
Al combinar **Laravel** en el backend con **React** en el frontend, obtenemos una arquitectura moderna, escalable y mantenible. Estas son las razones clave:

1. **Desacoplar responsabilidad:** Laravel gestiona datos, autenticación y rutas; React se centra en la interfaz y la interacción.
2. **Consumir APIs RESTful sin complicaciones:** Laravel crea endpoints listos para JSON y React los llama vía fetch o Axios.
3. **Acelerar el desarrollo:** Artisan y Eloquent automatizan el backend; componentes reutilizables y Vite optimizan el frontend.
4. **Garantizar seguridad y rendimiento:** Laravel ofrece validaciones, colas y protección CSRF; React optimiza el renderizado y la experiencia de usuario.

Así podemos iterar y escalar cada capa de forma independiente, manteniendo el código limpio y modular.

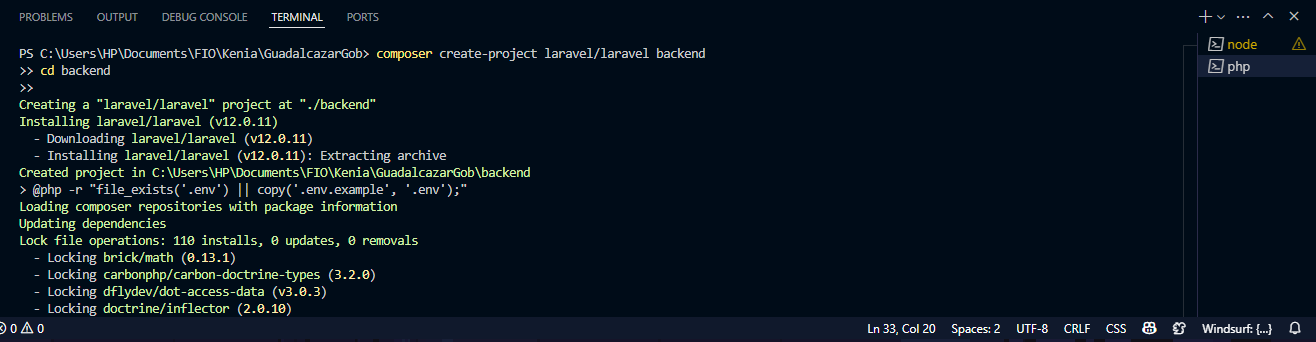
Flujo típico de trabajo

1. Definir modelos y migraciones → php artisan make:model Product -m.
2. Diseñar rutas y controladores → php artisan make:controller ProductController --api.
3. Construir y probar la API con Artisan (php artisan serve) y clientes HTTP (Thunder Client, Postman).
4. Crear vistas o consumir desde un frontend (React, Vue) usando JSON; Laravel gestiona la lógica y expone endpoints listos para consumir.

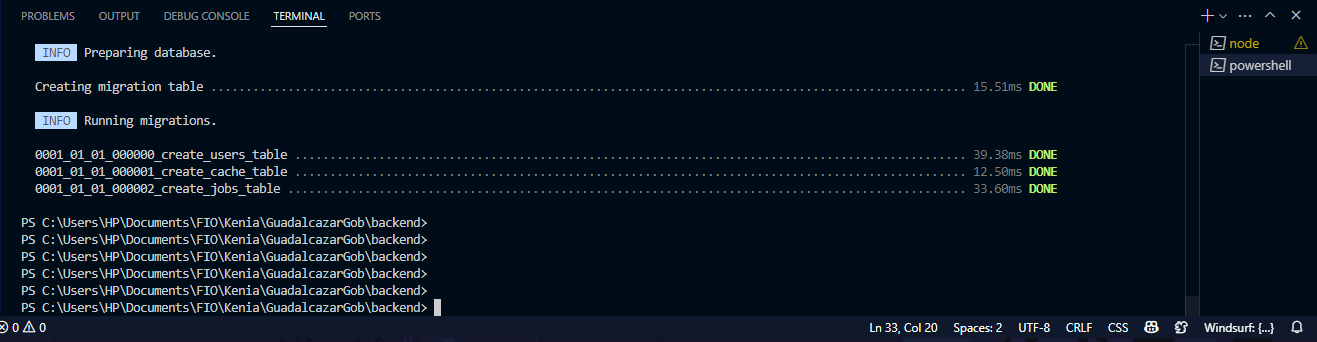
Con Laravel disponemos de un entorno sólido y probado para montar desde APIs sencillas hasta aplicaciones empresariales complejas, manteniendo siempre un equilibrio entre rapidez de desarrollo y buenas prácticas de ingeniería.

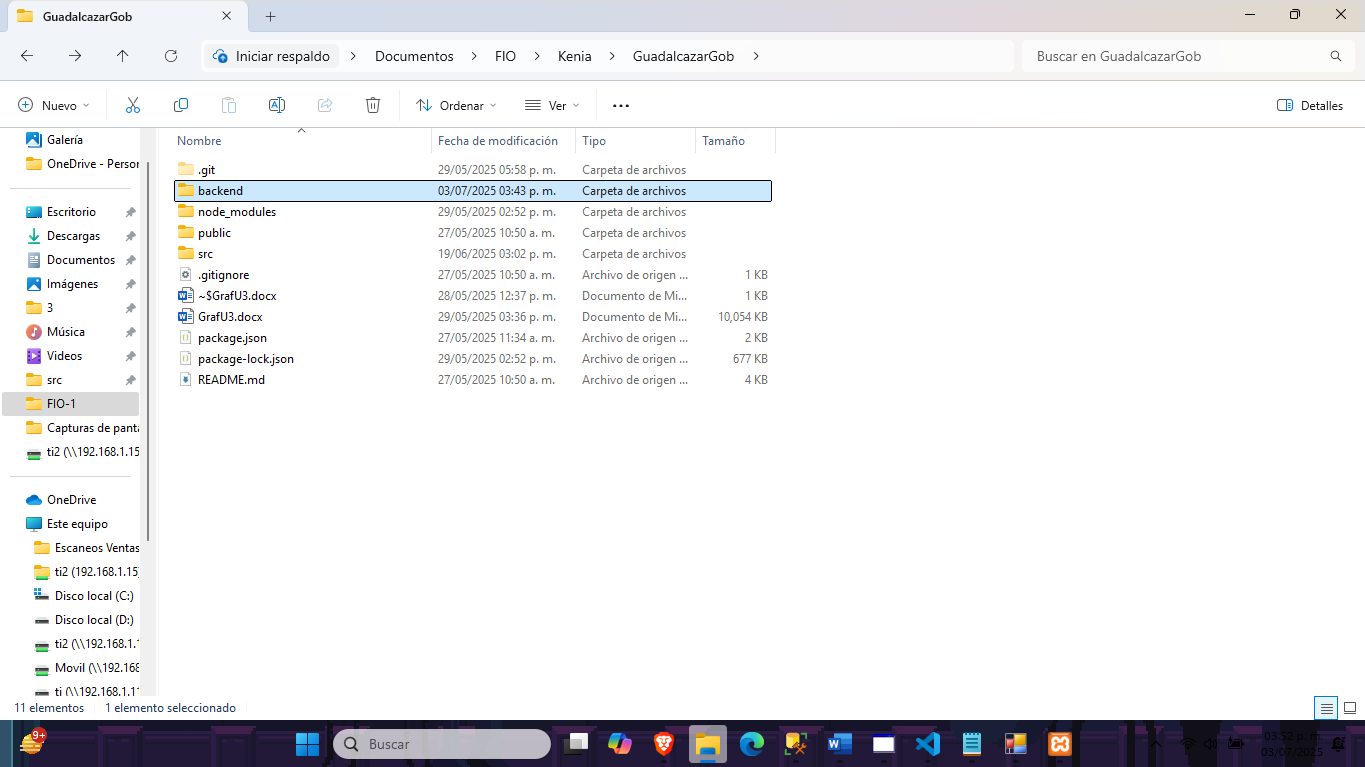
Instalación:

Instalamos Laravel para nuestro backend y creamos la bd llamada “guadalcazargob” desde MySQL.

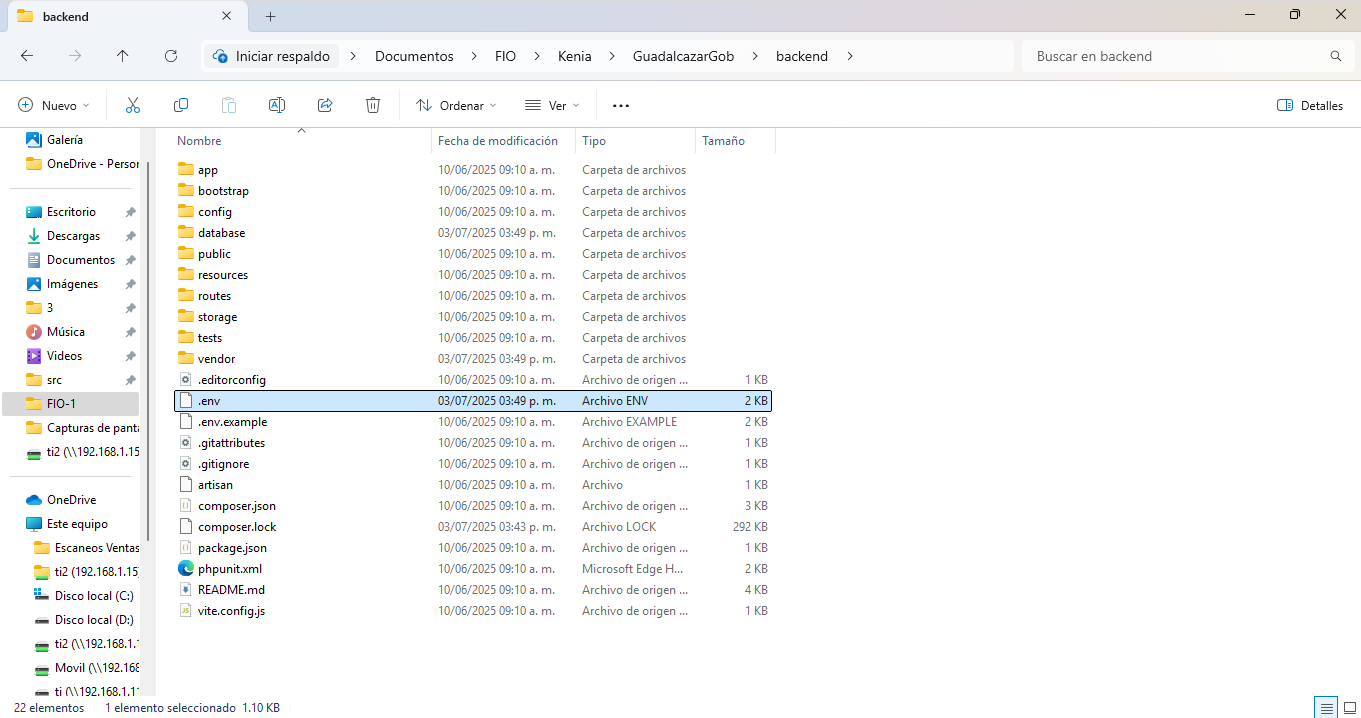


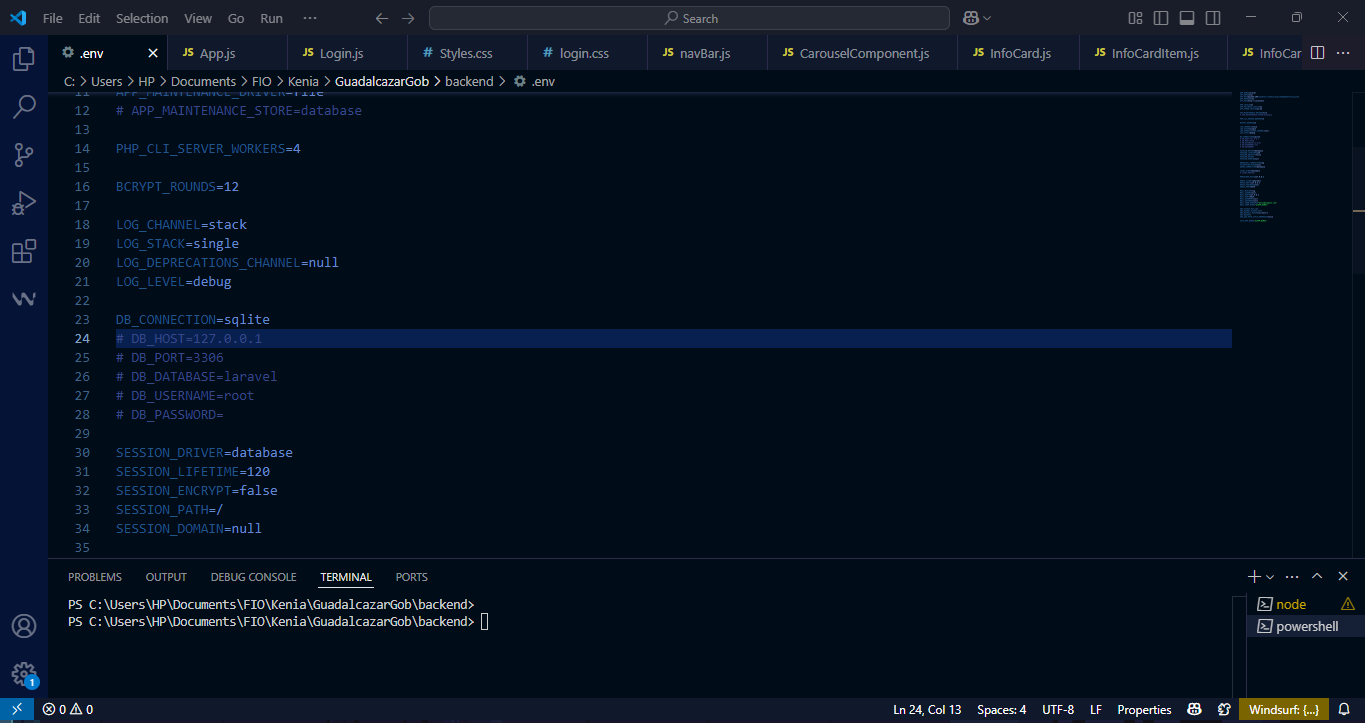
Cuando se haya terminado de instalar laravel debería aparecer algo asi

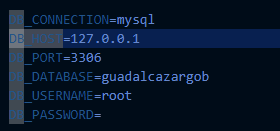


Y en la raíz de nuestro proyecto debería haberse creado una carpeta llamada “backend”  


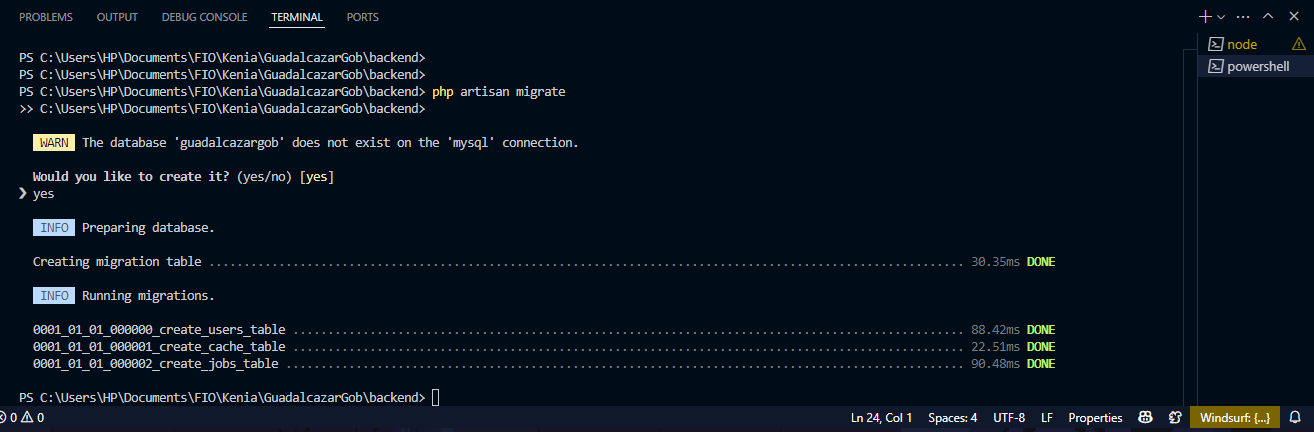
Ahora debemos modificar el archivo .env en la carpeta backend para poder enlazar nuestra base de datos a nuestro proyecto





Ajustamos estos parámetros para nuestras pruebas con XAMPP (Solamente son pruebas eventualmente agregaremos la seguridad y configuración pertinente)  


Y probamos si no hay algún error con el comando  
**#php artisan migrate**  
y si no da error como en la imagen entonces establecimos conexión exitosamente



Ahora creamos el endpoint para login

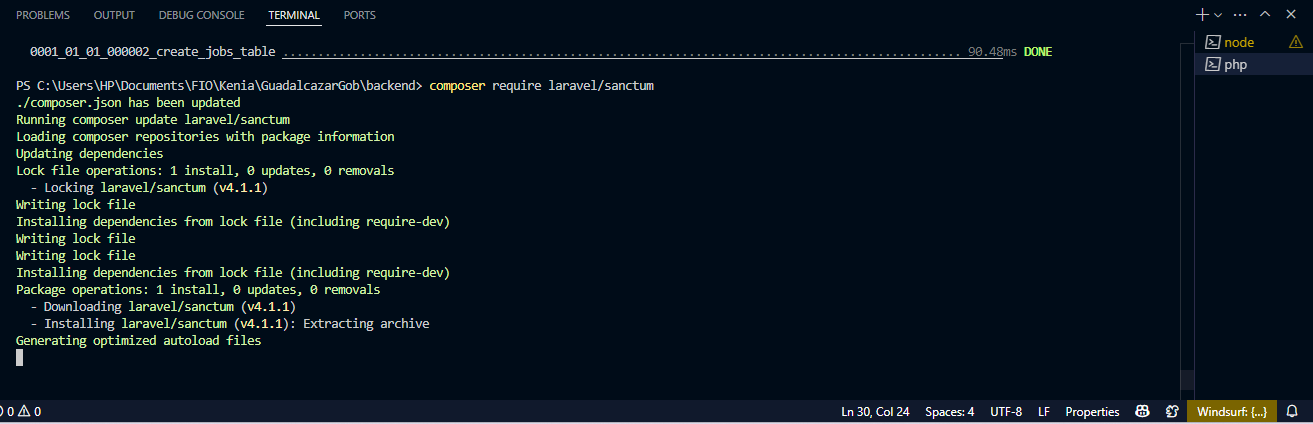
Vamos a usar el sistema de autenticación que trae Laravel (usando API tokens):

Instalamos con los siguientes comandos de Laravel Sanctum (para autenticar APIs, desde la carpeta raíz “backend”):

#composer require laravel/sanctum

#php artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"

#php artisan migrate



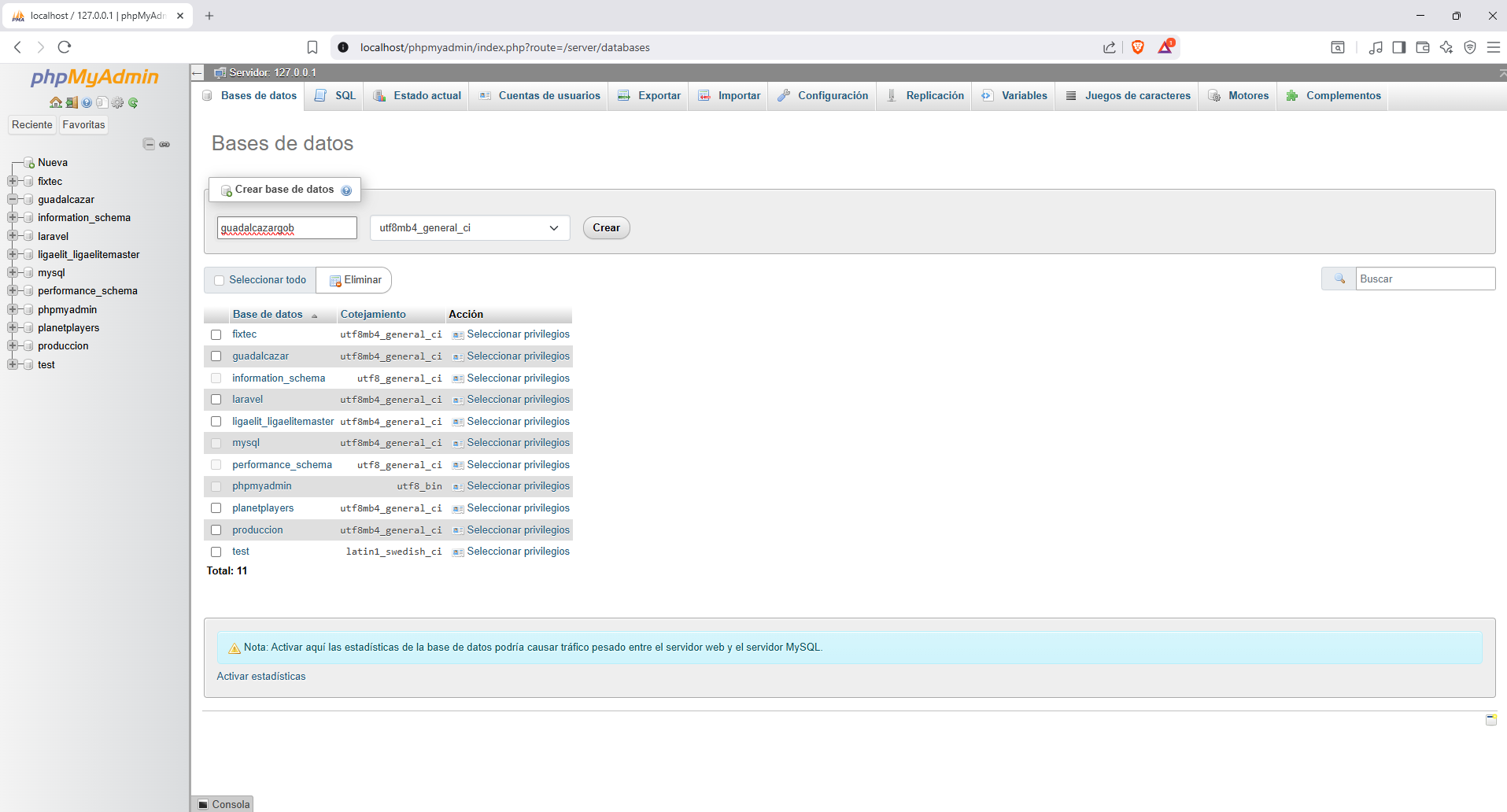
Para pruebas iniciales, usaremos XAMPP como entorno local: así instalamos Apache, MySQL y PHP de golpe, ejecutamos migraciones y seeders, y detectamos problemas de configuración o permisos de forma aislada.

Una vez validado el esquema y la lógica de datos, prepararemos el paso a producción en un servidor dedicado:

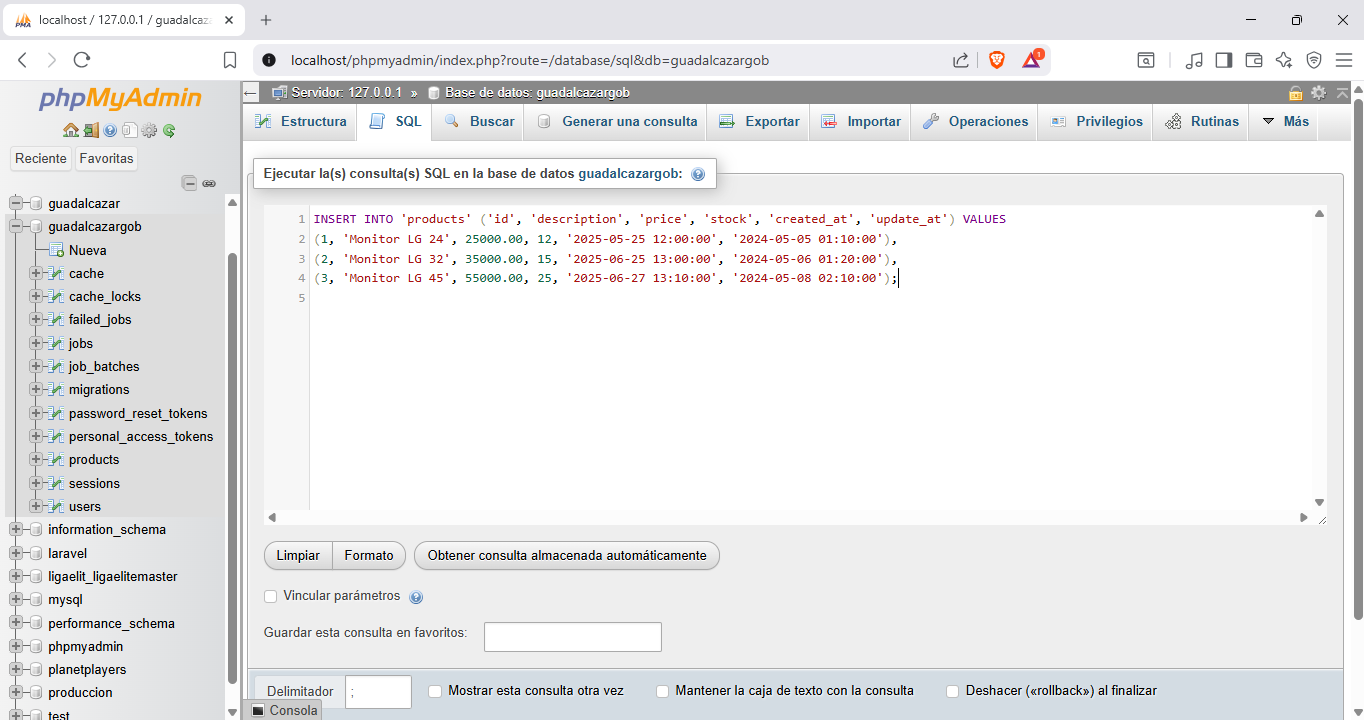
* **Exportar esquema y datos** (con mysqldump o migraciones automáticas).
* **Actualizar .env** para apuntar al nuevo host, puerto, usuario y contraseña.
* **Configurar seguridad:** restricciones de IP, cuentas con privilegios mínimos, backups programados y cifrado SSL/TLS.
* **Ajustar rendimiento:** parámetros de buffers, conexiones máximas y cacheo de consultas según carga prevista.

Este flujo —primero XAMPP en local, luego servidor dedicado— garantiza una migración suave de un entorno de pruebas ágil a uno productivo, robusto y seguro.

Aquí crearemos la base de datos de guadalcazargob para posteriormente enlazarla con nuestro backend.



Insertamos datos de prueba para probarlos desde un complemento de Visual Studio Code que en breve explicare.



Instalamos Thunder Client desde los complementos de visual studio code  
Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Para que usamos Thunder Client?

Por qué Thunder Client es una extensión ligera para Visual Studio Code diseñada para probar y depurar APIs HTTP/REST de forma rápida y sencilla, sin salir del editor.

Principales usos de Thunder Client

1. Enviar peticiones HTTP

* Soporta todos los métodos: GET, POST, PUT, PATCH, DELETE, etc.
* Permite configurar cabeceras, parámetros de ruta/query, cuerpo (JSON, form-data, x-www-form-urlencoded) y autenticación.

1. Organizar colecciones de peticiones

* Agrupa tus endpoints en carpetas o colecciones (por ejemplo, “Productos”, “Usuarios”, “Autenticación”).
* Guarda cada petición con su propio nombre y descripción.

1. Variables de entorno

* Define entornos (por ejemplo: local, staging, production) con variables como {{baseUrl}}, {{token}}, etc.
* Facilita alternar rápidamente de un servidor a otro sin editar cada URL manualmente.

1. Historial y respuestas

* Mantiene un registro de las peticiones enviadas, con fecha y hora.
* Visualiza respuestas en formatos:
* JSON (con resaltado de sintaxis y árbol plegable).
* Texto, HTML, o incluso vistas en bruto.
* Muestra tiempos de respuesta y tamaños de payload.

1. Autenticación simplificada

* Soporta esquemas comunes: Bearer Token, Basic Auth, API Key, OAuth2 (parcial).
* Guarda credenciales por entorno.

1. Tests y scripts (en versiones Pro)

* Permite definir assertions (comprobaciones) para automatizar validaciones de respuesta.
* Ejecutar pequeños snippets de JavaScript antes o después de la petición.

1. Integración con VS Code

* Sin cambiar de aplicación: todo ocurre dentro del mismo editor.
* Atajos de teclado para abrir y ejecutar tus peticiones.

¿Por qué usar Thunder Client?

* Ligero y rápido: ocupa pocos megas y consume menos recursos que herramientas más pesadas como Postman.
* Flujo más ágil: no necesitas cambiar de ventana, acelera el desarrollo al mantener todo en VS Code.
* Ideal para microservicios: cuando trabajas con APIs complejas o muchas rutas, te ayuda a mantenerlas organizadas.
* Gratuito y open source: la versión básica cubre la mayoría de necesidades sin costo.

A continuación, utilizaremos Thunder Client para validar que el backend responda correctamente: así podremos confirmar que cada endpoint existe, devuelve los datos esperados y maneja errores o validaciones como diseñamos. Una vez que comprobamos que la API funciona, pasamos a definir y ejecutar nuestras migraciones de Laravel para crear las tablas en la base de datos, revisando esquemas, relaciones y poblando datos de prueba con seeders o factories. Con la base de datos y la lógica del servidor ya validadas, el siguiente paso es integrar estas rutas con el frontend: construiremos componentes o páginas que consuman la API de forma dinámica, mostrando listados, formularios de creación/edición y vistas detalladas. De este modo seguimos un flujo incremental y modular—primero aseguramos el correcto funcionamiento de la capa de datos y negocio, y luego enlazamos esa lógica a las vistas, lo que nos permite detectar y corregir cualquier fallo de comunicación cliente-servidor antes de avanzar a la siguiente fase de diseño visual o interacción.

Ejemplo:

Con esta petición lo que hacemos es pedirle que nos traiga los datos que insertamos desde la bd.

Texto

El contenido generado por IA puede ser incorrecto.

Ahora le pediremos que cree un registro, cambiando el método por POST

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Nos da el status 200 de ok eso significa que si inserto los datos ahora lo que haremos es volver a enviar una solicitud GET para verificarlo visualmente.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Verificación visual:

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Y si como podemos ver esta agregado.

Ahora enviaremos una solicitud solo para que nos devuelva un solo id en este caso el 2 modificando la petición quitando {id} por el numero 2 que queremos.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Ahora la solicitud de eliminar

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Eliminaremos el id 4 que creamos al principio de nuestras pruebas

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.



Volvemos a traer todos los id para ver si fue eliminado correctamente

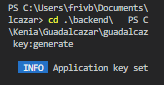
Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Para correr laravel correctamente necesitamos generar una key de lo contrario nos generara un error como este  
**Illuminate\Encryption\MissingAppKeyException**

**No application encryption key has been specified.**

Esto significa que Laravel no tiene configurada una clave de cifrado (APP\_KEY) en nuestro archivo .env



Al ejecutar este comando sucederán dos cosas:

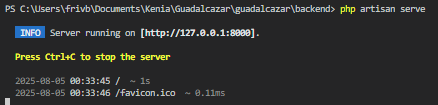
* **Generará una clave como: base64:largo\_valor\_aquí**
* **La insertará automáticamente en el archivo .env como APP\_KEY=...**

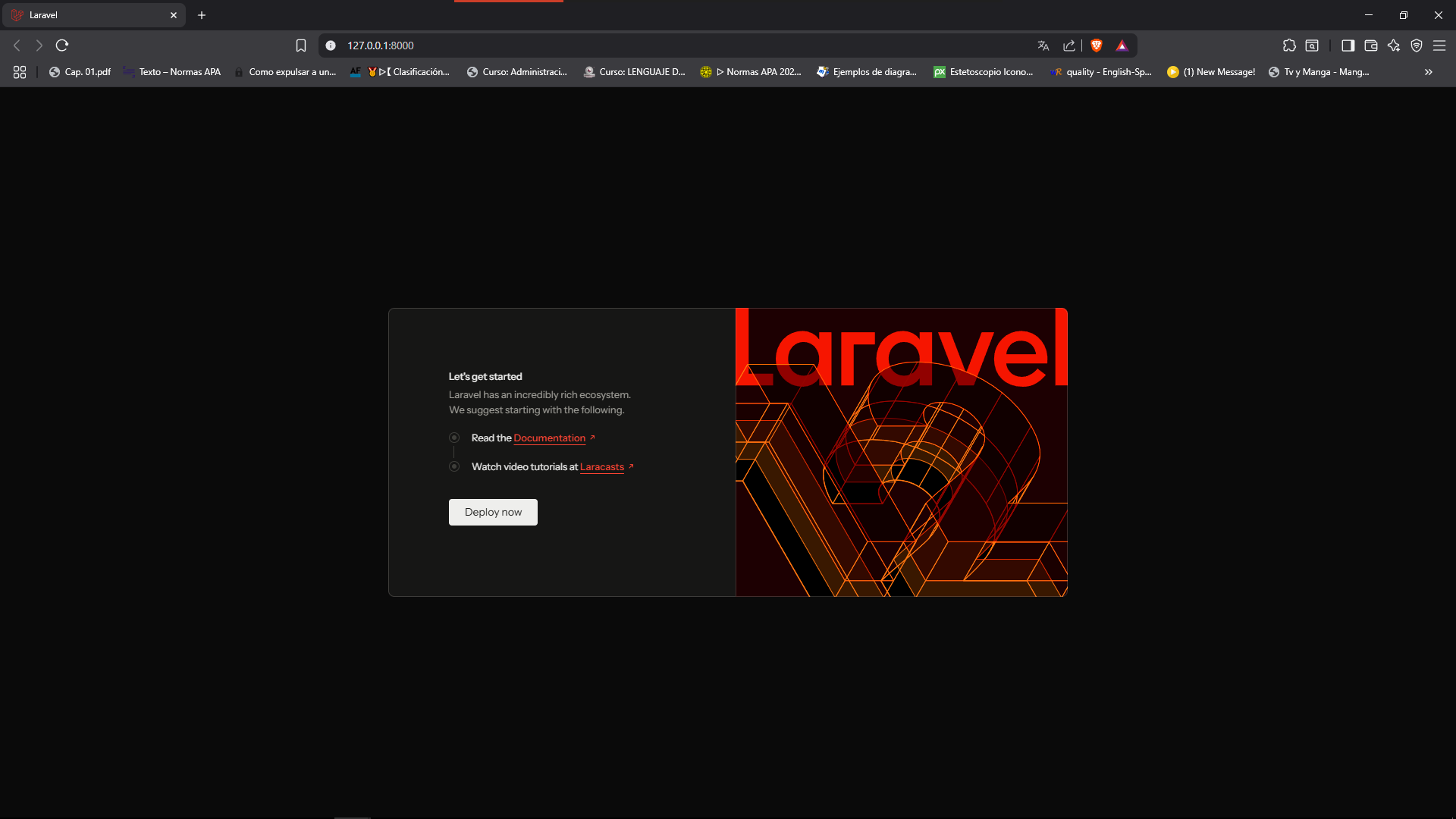
Laravel necesita esta clave para poder cifrar/de-cifrar datos como sesiones, cookies, etc.

Nunca debemos compartir la APP\_KEY en público si la aplicación ya está en producción pero aquí pondremos la que se genero ya que es parte del proyecto y de igual manera se podría generar una nueva mas adelante si hubiera algún inconveniente

**APP\_KEY=base64:eKgXuGWvCmgH85n0Frp5Pmnm3pj/8als5VdtlOoa4yI=**

Y después



Para correr el servidor y en el navegador web nos debe aparecer algo asi  


Ese comando **debe estar activo** en la consola mientras desarrollamos de lo contrario nos aparecerá un mensaje como “No se puede acceder a este sitio”.

Ahora utilizaremos



**Git y GitHub**

Durante el desarrollo de este proyecto, utilicé **Git** y **GitHub** como herramientas fundamentales para el control de versiones, la gestión del código fuente y la organización del flujo de trabajo. Estas herramientas me permitieron mantener un historial detallado de todos los cambios realizados, asegurar la integridad del proyecto, y facilitar la recuperación ante errores o fallos inesperados.

**Git** es un sistema de control de versiones distribuido, ampliamente utilizado en el desarrollo de software. Su principal ventaja es que permite registrar y administrar cambios en el código a lo largo del tiempo, lo que resulta especialmente útil cuando se trabaja en proyectos de mediano o largo plazo. Gracias a Git, pude documentar cada modificación relevante mediante *commits*, lo que me permitió tener un control preciso sobre el avance del proyecto, entender qué se cambió, cuándo y por qué.

Por otro lado, utilicé **GitHub** como plataforma para alojar el repositorio remoto del proyecto. GitHub no solo me permitió almacenar mi código en la nube, sino también contar con una copia de seguridad centralizada y segura. Esto fue muy útil para acceder al proyecto desde diferentes dispositivos y para asegurar que el trabajo no se perdiera en caso de una falla en mi equipo local. Además, GitHub ofrece herramientas visuales para revisar el historial de commits, comparar versiones de archivos y gestionar ramas de desarrollo.

Para ejecutar los comandos de Git utilicé **Git Bash**, una terminal que me permitió interactuar con Git desde la línea de comandos. A través de esta herramienta realicé acciones como:

* **Inicializar el repositorio** con git init.
* **Subir el proyecto** por primera vez a GitHub usando git remote add origin y git push -u origin master.
* **Registrar cambios** con git add y git commit -m.
* **Actualizar mi repositorio local** con los últimos cambios usando git pull.
* **Explorar el historial de cambios** con git log y git diff.

Además, una de las funcionalidades más importantes que utilicé fue la posibilidad de **crear ramas (branches)**. Esto me permitió trabajar en nuevas características o probar soluciones alternativas sin afectar el código principal. Por ejemplo, cuando quise implementar una nueva funcionalidad que no estaba seguro si funcionaría correctamente, creé una rama separada, realicé las pruebas correspondientes y, una vez validado que todo funcionaba bien, integré los cambios al código principal mediante git merge.

En ocasiones, cuando una modificación causaba errores o fallos en el sistema, Git me permitió **volver fácilmente a una versión anterior estable** utilizando comandos como git checkout o git reset. Esta capacidad fue fundamental para evitar pérdida de tiempo o retrocesos mayores, y me dio la tranquilidad de poder experimentar sin comprometer el proyecto completo.

Además de todas estas funciones, Git también me ayudó a mantener un flujo de trabajo más organizado y profesional. Cada cambio quedó documentado y justificado en el historial, lo cual facilita el mantenimiento del código y la comprensión del mismo a futuro.

El uso de Git y GitHub fue esencial para el desarrollo eficiente y ordenado de este proyecto. Aunque fui el único desarrollador involucrado, estas herramientas me ofrecieron grandes ventajas como control de versiones, respaldo automático, gestión de ramas y recuperación ante errores. Aprender a utilizarlas correctamente no solo me permitió mejorar mi productividad, sino que también fortaleció mis habilidades como desarrollador, preparándome para futuros trabajos en equipo y proyectos más complejos.

Aquí algunos ejemplos:

Aquí utilizamos para crear el inicio de sesión ya que íbamos a modificar cosas muy sensibles del código y queríamos probar sin tener miedo a corromper el proyecto  
