5. Conclusiones

5.1. Resultados Obtenidos

El desarrollo de Agatha ha dado como resultado una API sólida, estructurada y funcional, pensada para ofrecer una experiencia fluida en una aplicación creativa centrada en la escritura.
Se han implementado todas las características principales planteadas al inicio:

- Sistema de autenticación seguro con Laravel Sanctum.
- Gestión completa de historias con validaciones, aleatoriedad y formato.
- Generación automática de palabras/lugares aleatorios para fomentar la creatividad.
- Sistema automatizado de notificaciones por inactividad mediante comandos programados.
- Estructura clara de rutas y controladores que permite escalar funcionalidades.

En conjunto, Agatha es una base estable para una aplicación real.

5.2. Aprendizajes Valiosos

Este proyecto ha supuesto un salto importante en mis conocimientos tanto de backend como de frontend, además de en herramientas de desarrollo reales que no había utilizado antes. A lo largo del desarrollo he aprendido:

Laravel y desarrollo backend

- Gestión de autenticación con **Laravel Sanctum** (tokens personales, middleware `auth:sanctum`).
- Construcción de **Mailables**, plantillas Markdown y envío de correos desde la app.
- Programación de tareas automáticas (**Scheduler**).
- Creación y ejecución de **Console Commands**, y cómo integrarlos con lógica real del proyecto.

Vue 3 + Vite (Frontend)

Empece el proyecto sin haber trabajado antes con Vue, y ha sido una de las partes donde más he aprendido:

- Uso de **Composition API** (`ref`, `reactive`, `onMounted`, etc.).
- Manejo de **Pinia** como store global para:
  - guardar el usuario autenticado,
  - guardar y refrescar el token,
  - gestionar sesiones caducadas.

- Router de Vue y protección de rutas mediante **guards** basados en meta `requiresAuth`.
- Envío de peticiones con Axios e interceptores para añadir el token automáticamente.
- Validación de formularios y visualización de errores provenientes del backend.
- Experiencia real con comunicación asíncrona con la API.

Infraestructura local y herramientas nuevas

Este proyecto también me ha permitido aprender varias herramientas que nunca había usado:

- **Laragon**
  - Instalación, configuración y uso como entorno de desarrollo.
  - Manejo del host virtual automático (`agatha-api.test`).
- **Mailtrap**
  - Configuración SMTP para pruebas de envío de emails.
  - Lectura y depuración de correos sin afectar a usuarios reales.

- **Scheduler y Cron**
  - Ejecución de tareas recurrentes con `php artisan schedule:work`.
  - Cómo funcionan los crons reales en servidores Linux.
- **Comandos personalizados**
  - Crear tareas que se ejecutan desde CLI con `php artisan`.
  - Integrarlos con el scheduler.

Gestión de errores y depuración

- Leer y entender errores de Laravel y Vue.
- Depurar respuestas 422 de validación.
- Manejo de excepciones con `try/catch` y mensajes JSON claros.
- Aprendí la importancia de devolver siempre un formato consistente desde el backend.

Git y documentación

- Organización del proyecto con Git/GitLab.
- Creación de documentación en **Markdown**, incluyendo diagramas, tablas, ejemplos y fragmentos de código.

En conjunto, este proyecto me ha permitido aprender cómo se construye una aplicación completa de principio a fin: desde la base de datos, pasando por la API, hasta la interfaz web final que consume esa API.

Además de lo técnico, también ha sido un ejercicio de planificación y resolución de problemas, especialmente al depurar comportamientos inesperados con fechas, validaciones o límites de servicio.

5.3. Cosas que haría de otra manera

Con lo aprendido durante el desarrollo, hay ciertos puntos que replantearía si empezara de cero:

- **Implementar pruebas automatizadas** para evitar errores en funciones sensibles como el scheduler o los mailables.
- **Introducir un sistema de colas (queues)** para el envío de correos, evitando bloqueos por límites de envío.

- **Definir la arquitectura de base de datos desde el principio**, incluyendo tablas auxiliares que se añadieron más tarde.

Cada uno de estos cambios facilitaría el mantenimiento a largo plazo.

5.4. Posibles Mejoras Futuras

El proyecto ha quedado bien definido, pero aun puede mejorarse:

- Añadir un panel estadístico para que el usuario vea su actividad (días consecutivos, número de historias, etc.).

- Implementar etiquetas, categorías o filtros para organizar historias.
- Permitir compartir historias mediante enlaces públicos.
- Incluir un modo competitivo (reto diario, rankings, etc.).
- Autoguardado de la historia en borradores mientras que no se pulse en guardar para no perder el trabajo nunca.
- Boton para mostrar contraseña al hacer login, cambio de contraseña y registro.
- Modo oscuro/claro del sitio web.

5.5. Continuidad y Planes

Una vez finalizado este proyecto, las opciones de continuidad son amplias:

- Integrar un editor de texto con estilos, emoticonos, etc.
- Preparar un despliegue en producción (Railway, Render, Forge...).
- Convertirlo en un proyecto personal a largo plazo para practicar nuevas tecnologías.

5.6. Despliegue y Entorno de Producción

Aunque el proyecto se ha desarrollado y probado principalmente en entorno local con Laragon, se ha preparado la base de datos para un futuro despliegue en producción utilizando **Railway** como plataforma cloud.

Migración de la base de datos a Railway

- Se exportó la base de datos completa desde phpMyAdmin (incluyendo estructura, datos de prueba, usuarios, historias, palabras y lugares).

- Se importó exitosamente en un servicio MySQL de Railway.
- La conexión remota desde el entorno local de Laravel se configuró correctamente modificando las variables de entorno en `.env`:

```
DB_CONNECTION=mysql
DB_HOST=[host Railway]
DB_PORT=[puerto Railway]
DB_DATABASE=[nombre BBDD]
DB_USERNAME=[usuario]
DB_PASSWORD=[contraseña]
```

Se verificó la conexión correcta ejecutando comandos como:

```
php artisan config:clear
php artisan cache:clear
php artisan tinker
```

Y dentro de tinker:

```
DB::connection()->getPdo();
```

El resultado confirmó la conexión exitosa a través del proxy de Railway (maglev.proxy.rlwy.net).

Ventajas de esta configuración

- Permite trabajar en desarrollo local contra datos reales en la nube (usuarios de prueba, historias existentes, estado de inactividad).
- Facilita pruebas más realistas del sistema de avisos por inactividad y del comportamiento general de la aplicación.
- Prepara el terreno para un despliegue completo futuro: el backend Laravel puede subirse a Railway. El frontend Vue podría desplegarse en Vercel.