1. Introducción

Este documento presenta la evidencia de desarrollo del sistema **AMCA (Apoyo al Mercado Campesino)**, una aplicación web construida con **Laravel 12 (PHP 8.2+)** que integra módulos para la gestión de recursos agrícolas (animales, vegetales, preparados, agricultores y fincas), tanto desde una interfaz web como mediante una **API REST**. Se incluyen diagramas, pruebas, documentación de API y evidencias de funcionamiento.

2. Requerimientos del Sistema

- Gestión CRUD de: animales, vegetales, preparados, agricultores y fincas.
- Búsqueda y visualización pública de elementos.
- Subida de imágenes para cada recurso.
- Autenticación para el área administrativa.
- API REST con endpoints para integración externa.

3. Historias de Usuario / Casos de Uso

- **Como visitante**: quiero buscar y ver detalles de productos agrícolas.
- **Como administrador**: quiero gestionar (crear, editar, eliminar) animales, vegetales, preparados, agricultores y fincas.
- **Como consumidor de API**: quiero acceder a los datos en formato JSON.

4. Entorno de Desarrollo

- **IDE**: VS Code con extensión Postman.
- **Backend**: Laravel 12, PHP 8.2+, Composer.
- **Frontend**: Vite, Bootstrap 5, Tailwind.
- **Base de datos**: SQLite (por defecto) o MySQL.
- **Control de versiones**: Git con flujo de ramas feature/*, release/*, hotfix/*.

5. Diagramas de Arquitectura y Diseño

5.1. Diagrama de Clases
Relaciones entre entidades del dominio agrícola.
5.2. Diagrama de Componentes

5.3. Diagrama de Secuencia (Crear Animal)

Flujo de creación de un animal mediante API.
5.4. Diagrama de Flujo
Proceso de subida de imágenes en la API.

6. Mecanismos de Segurida

- Autenticación con laravel/ui y middleware auth.
- Protección CSRF en formularios web.
- Validación de entrada en controladores y APIs.
- Sanitización de archivos subidos (imágenes).

7. Estructura de Capas y Componentes

Capa	Componentes
Presentación	Blade, Vite, Bootstrap, Tailwind
Aplicación	Controladores, Rutas, Middleware
Dominio/Datos	Modelos Eloquent, Migraciones

8. Metodología de Desarrollo

- **Enfoque**: Iterativo-incremental.
- **Control de versiones**: Git con convención de commits (feat:, fix:, docs:, etc.).
- **Pruebas**: PHPUnit para pruebas unitarias y funcionales.
- **CI/CD**: Preparado para integración continua (ej: GitHub Actions).

9. Mapa de Navegación

- /pagina_home → Página principal
- /pagina_resultados → Búsqueda
- /pagina_detalle/{id}/{tipo} → Detalle
- /animales, /vegetales, ... → CRUD administrativo
- /api/* → Endpoints REST

10. Codificación y Buenas Prácticas

- **Patrón MVC**: Separación clara de responsabilidades.
- **API RESTful**: Uso de apiResource y respuestas JSON estandarizadas.
- **Validaciones**: Form Requests y validadores en controladores.
- **Subida de archivos**: Almacenamiento en public/img/ con nombres únicos.
- **Pruebas automatizadas**: Cubren CRUD y subida de imágenes.

11. Control de Versiones

- Repositorio Git con ramas: main, feature/*, hotfix/*.
- Convención de commits:
  - feat: para nuevas funcionalidades
  - fix: para correcciones
  - docs: para documentación
  - test: para pruebas

12. Librerías y Frameworks

Capa	Tecnologías
Frontend	Bootstrap, Tailwind, Vite
Backend	Laravel, Eloquent, Validator
Base de datos	SQLite / MySQL
Pruebas	PHPUnit, Postman

13. Pruebas Unitarias y Funcionales

- **Ubicación**: tests/Feature/ y tests/Unit/.
- **Comando**: php artisan test.
- **Cobertura**:
  - CRUD de entidades
  - Subida de imágenes
  - Validaciones
  - Códigos HTTP

14. Configuración de Servidores y BD

- **Desarrollo**:
  - php artisan serve
  - npm run dev
- **Producción**:
  - APP_ENV=production
  - APP_DEBUG=false
  - Cache de rutas, vistas y configuraciones

15. Documentación de API

- Colecciones Postman incluidas en docs/:

- AMCA_Backend.postman_collection.json
- AMCA_REST.postman_collection.json
- Ejemplos de requests en cURL y JSON.

16. Evidencias de Ejecución
- Pruebas de API con Postman (VS Code).
- Resultados de PHPUnit: **14 tests, 54 assertions, OK**.
- Imágenes subidas y almacenadas en public/img/.
- Registros en base de datos verificados.

17. Conclusiones
El sistema **AMCA** cumple con los requerimientos de integración de módulos para una aplicación web, utilizando arquitectura MVC, API REST, pruebas automatizadas y documentación completa. Está listo para despliegue en entornos de producción y permite escalabilidad para futuras funcionalidades.

18. Anexos
- README.md - Instalación y configuración.
- API_Testing_Evidence.md - Pruebas de API.
- project_audit_2025-09-01.md - Auditoría de código.
- Diagramas en formato SVG y MD.