

# Arquitectura del Sistema HybridSecScan

## Documentación UML para Análisis de Seguridad Híbrido (SAST + DAST)

**Autor:** Oscar Isaac Laguna Santa Cruz

**Institución:** Universidad Nacional Mayor de San Marcos

**Fecha:** Noviembre 2025

**Versión:** 1.0

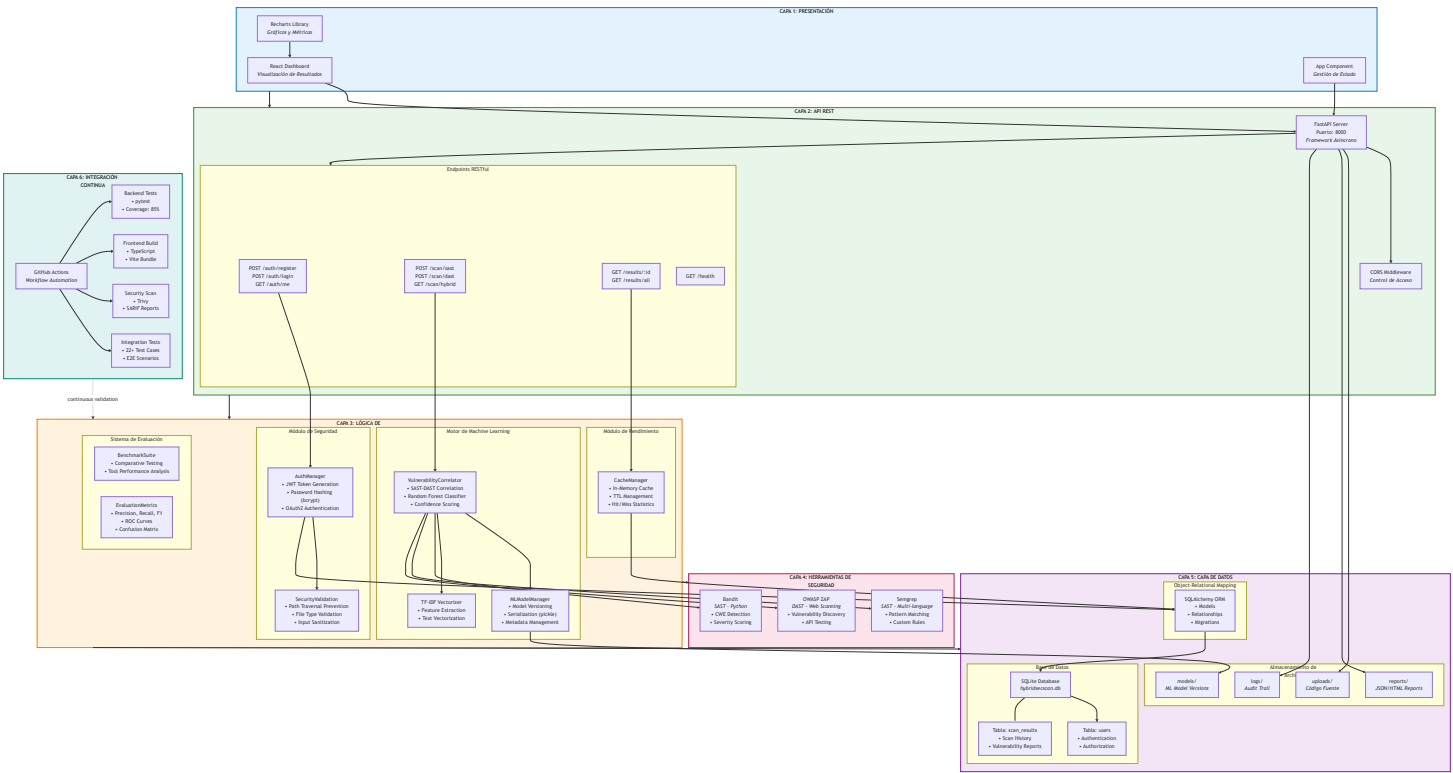
## Índice

1. [Vista General del Sistema](#)
2. [Diagrama de Clases](#)
3. [Diagramas de Secuencia](#)
4. [Diagrama de Componentes](#)
5. [Diagrama de Estados](#)
6. [Diagrama de Despliegue](#)
7. [Estructura de Paquetes](#)
8. [Patrones y Principios](#)

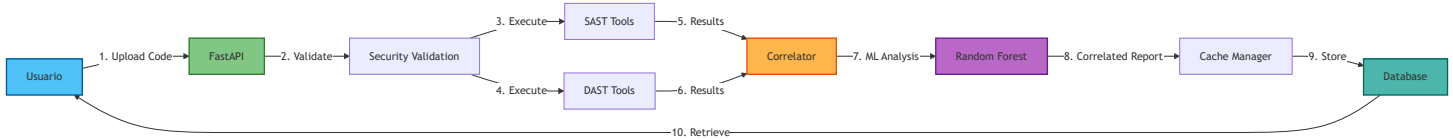
## 1. Vista General del Sistema

### Arquitectura en Capas (Layered Architecture)

El sistema HybridSecScan implementa una arquitectura en capas de 6 niveles que garantiza la separación de responsabilidades y facilita el mantenimiento y escalabilidad del sistema.



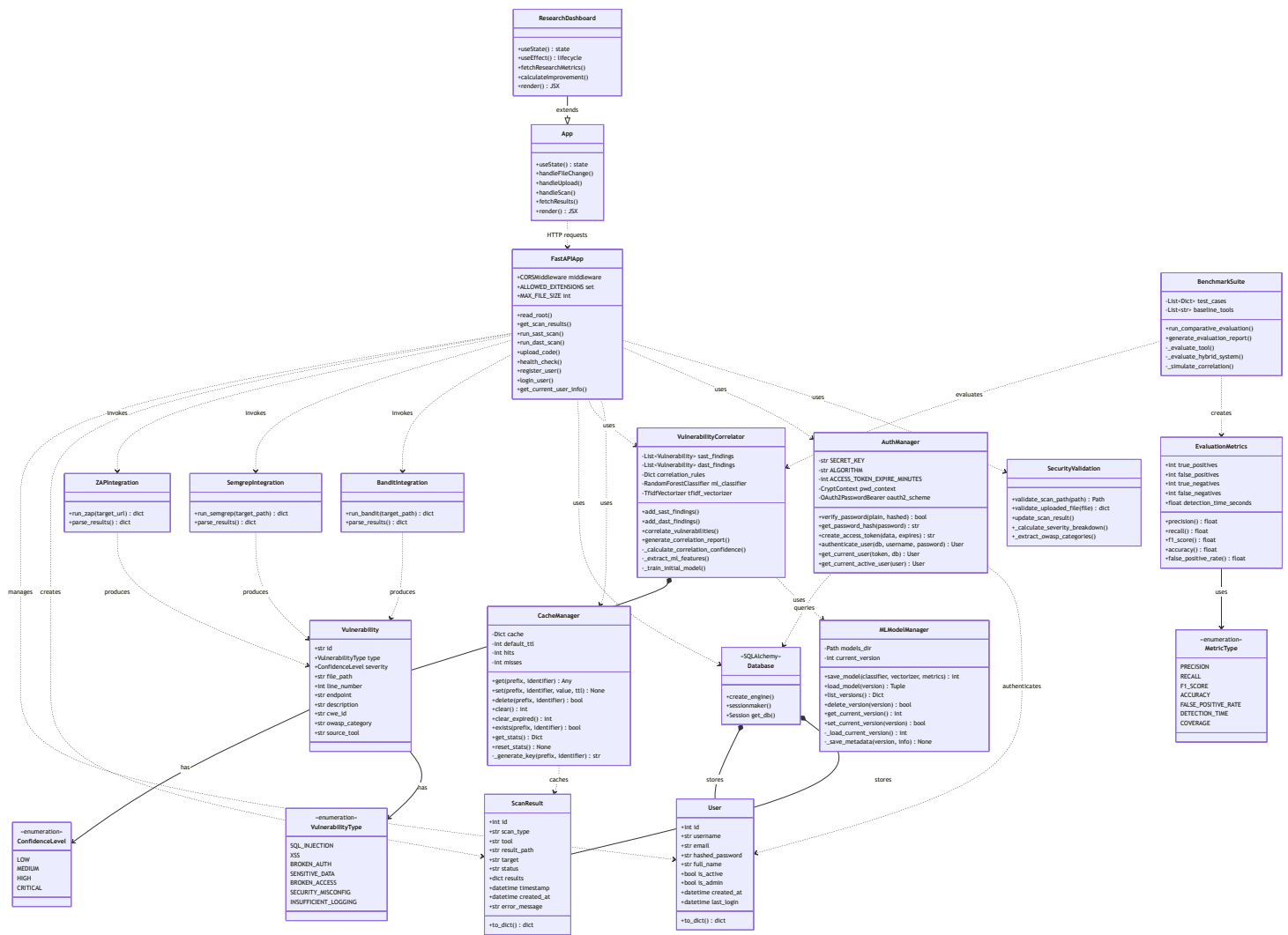
## Flujo de Datos Principal



## 2. Diagrama de Clases

### Modelo Completo del Sistema

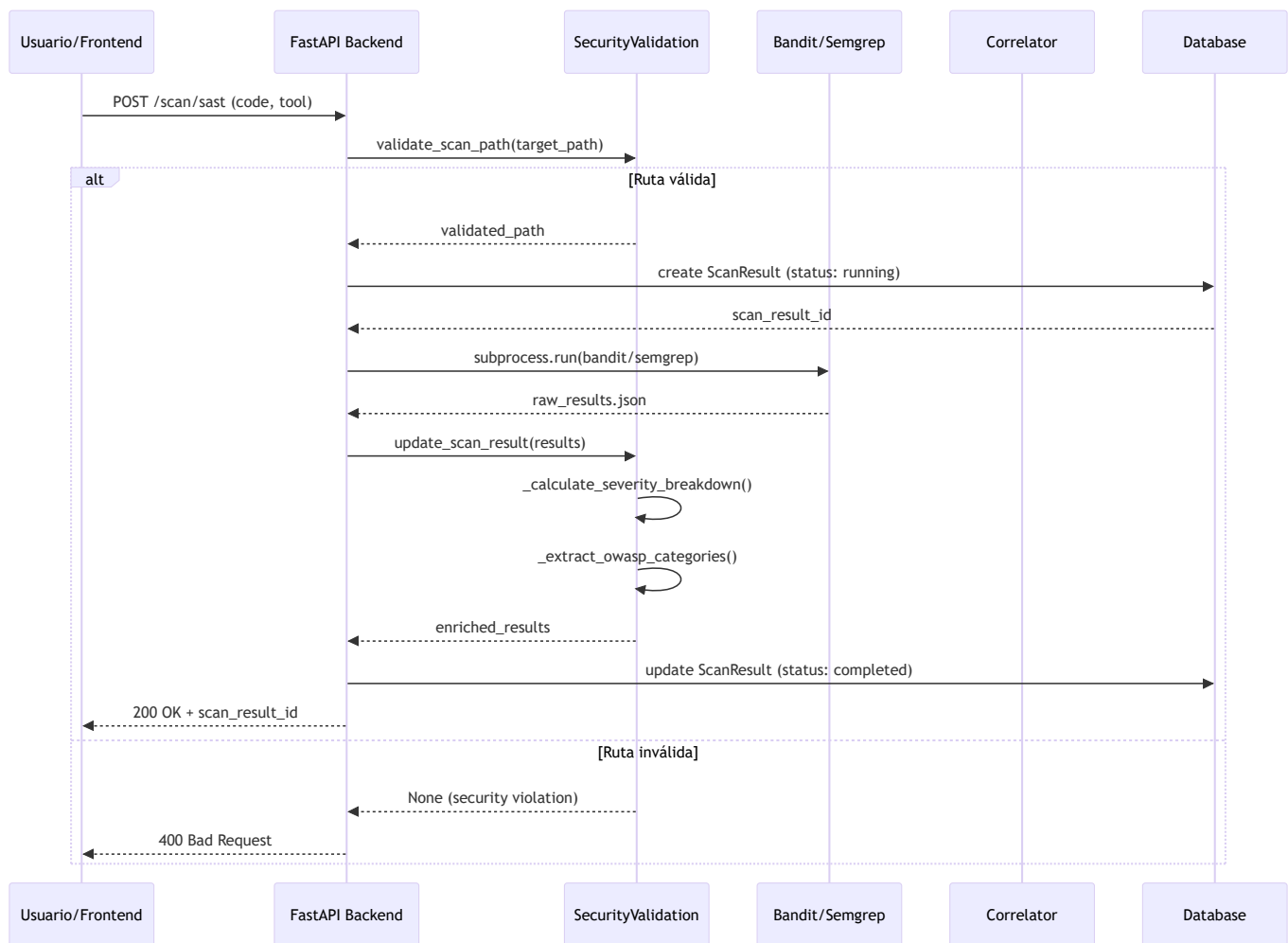
El siguiente diagrama de clases presenta la estructura completa del sistema, incluyendo todas las clases principales, sus atributos, métodos y relaciones. Este modelo sigue los principios SOLID y patrones de diseño reconocidos en la industria.



## 3. Diagramas de Secuencia

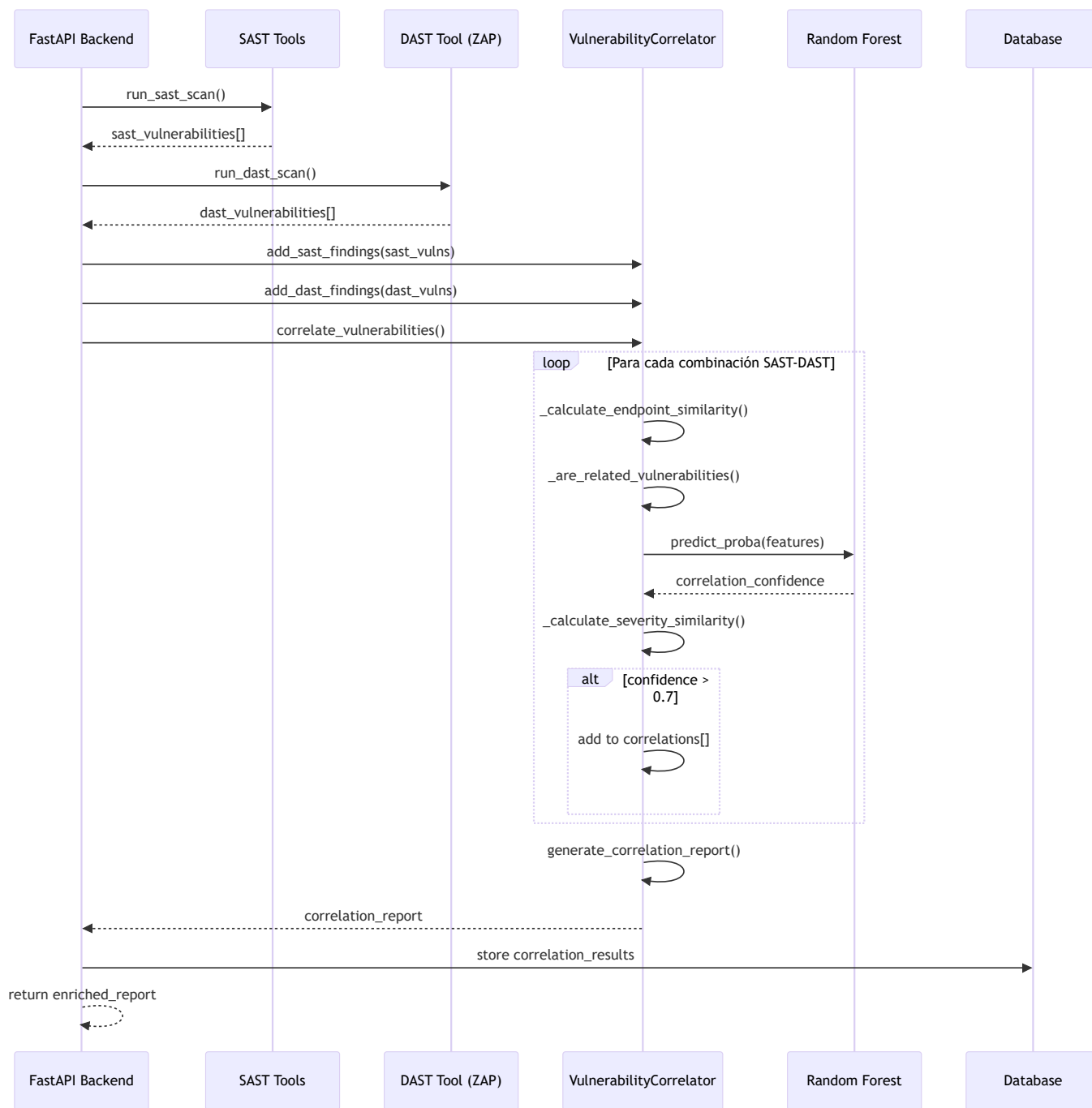
### 3.1. Flujo de Análisis SAST (Static Application Security Testing)

Este diagrama muestra la interacción temporal entre los componentes del sistema durante la ejecución de un análisis de seguridad estático. Se incluyen validaciones de seguridad, procesamiento de resultados y almacenamiento persistente.



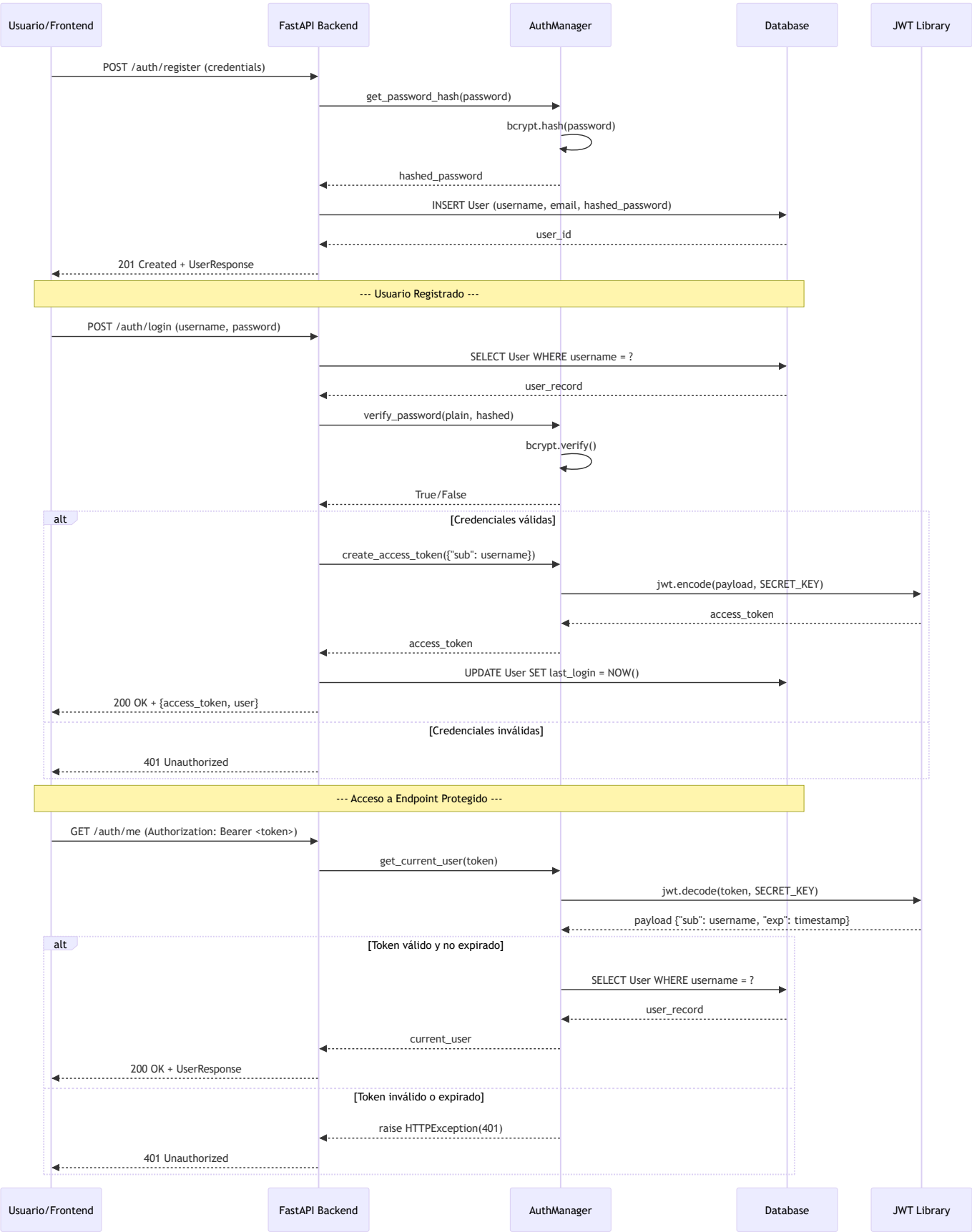
## 3.2. Flujo de Correlación Híbrida (SAST + DAST)

Este diagrama ilustra el proceso de correlación inteligente entre vulnerabilidades detectadas por herramientas SAST y DAST, utilizando algoritmos de Machine Learning (Random Forest) para determinar la confianza de correlación y reducir falsos positivos.



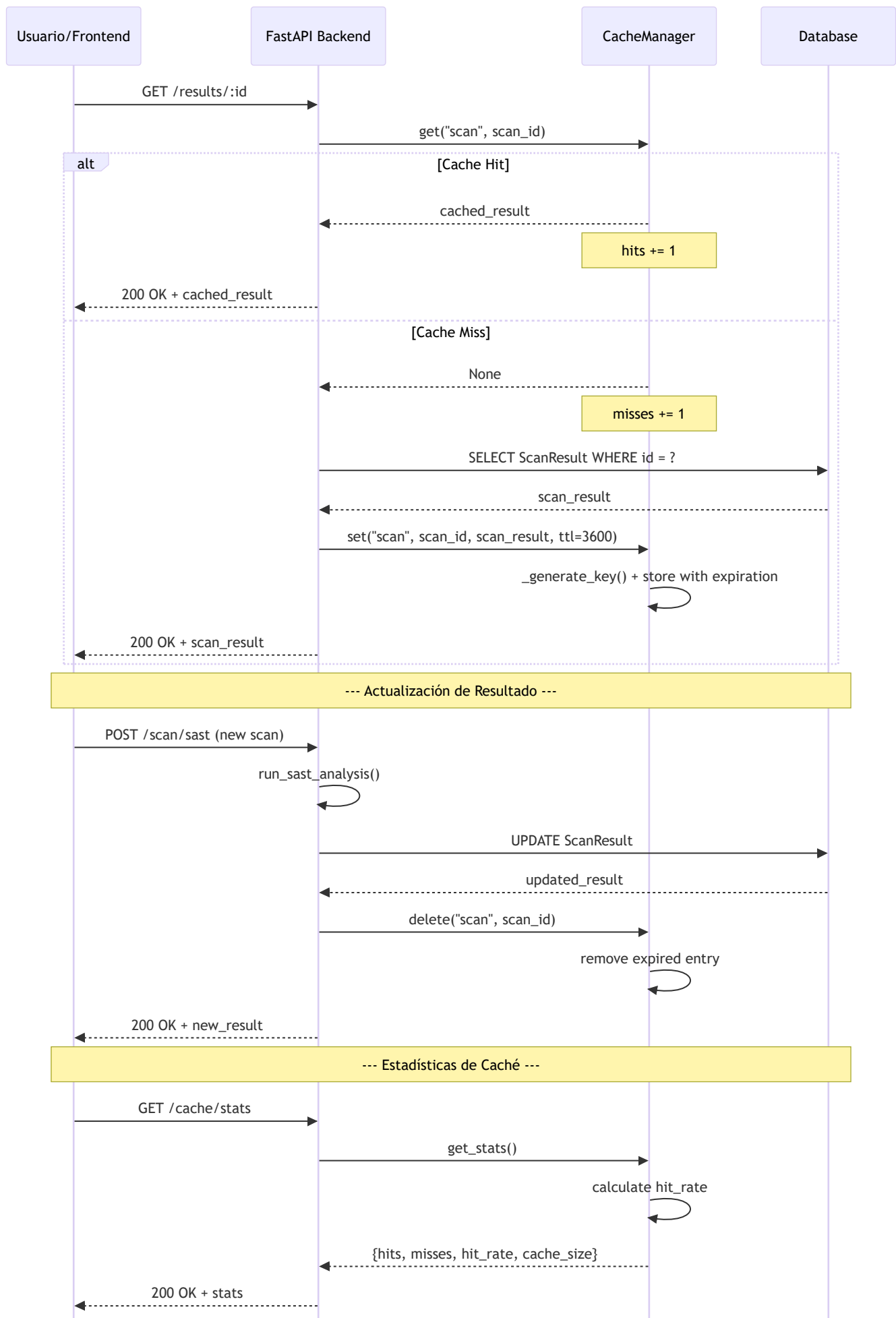
### 3.3. Sistema de Autenticación y Autorización (JWT)

Implementación completa del ciclo de autenticación utilizando JSON Web Tokens (JWT) con hashing bcrypt para contraseñas. Incluye registro de usuarios, login, validación de tokens y acceso a recursos protegidos.



### **3.4. Optimización de Rendimiento - Sistema de Caché**

Estrategia de caché con Time-To-Live (TTL) para reducir latencia en consultas repetitivas. Implementa patrón Cache-Aside con estadísticas de hit rate para monitoreo de eficiencia.



Usuario/Frontend

FastAPI Backend

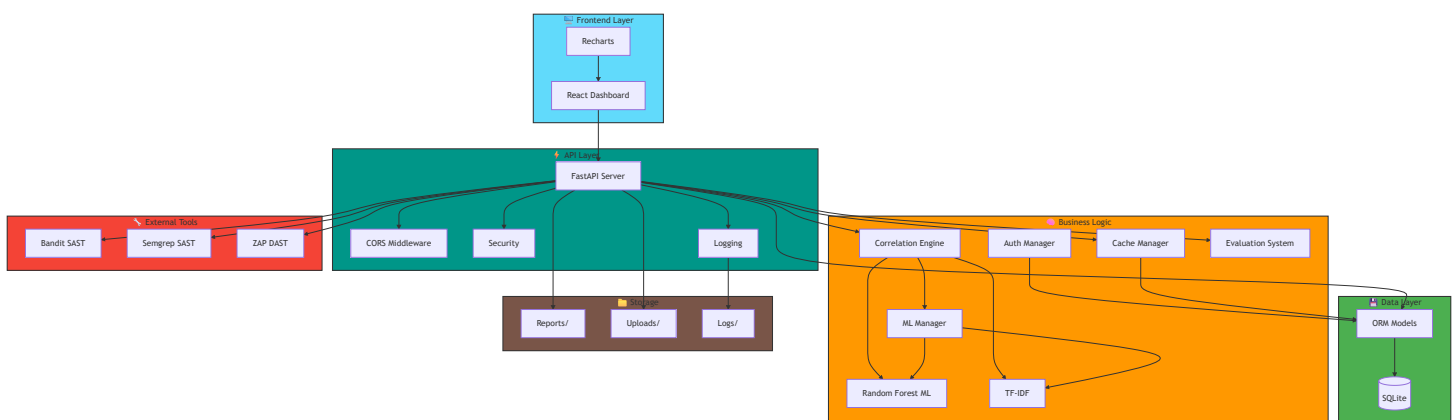
CacheManager

Database

## 4. Diagrama de Componentes

### Vista de Componentes y Dependencias

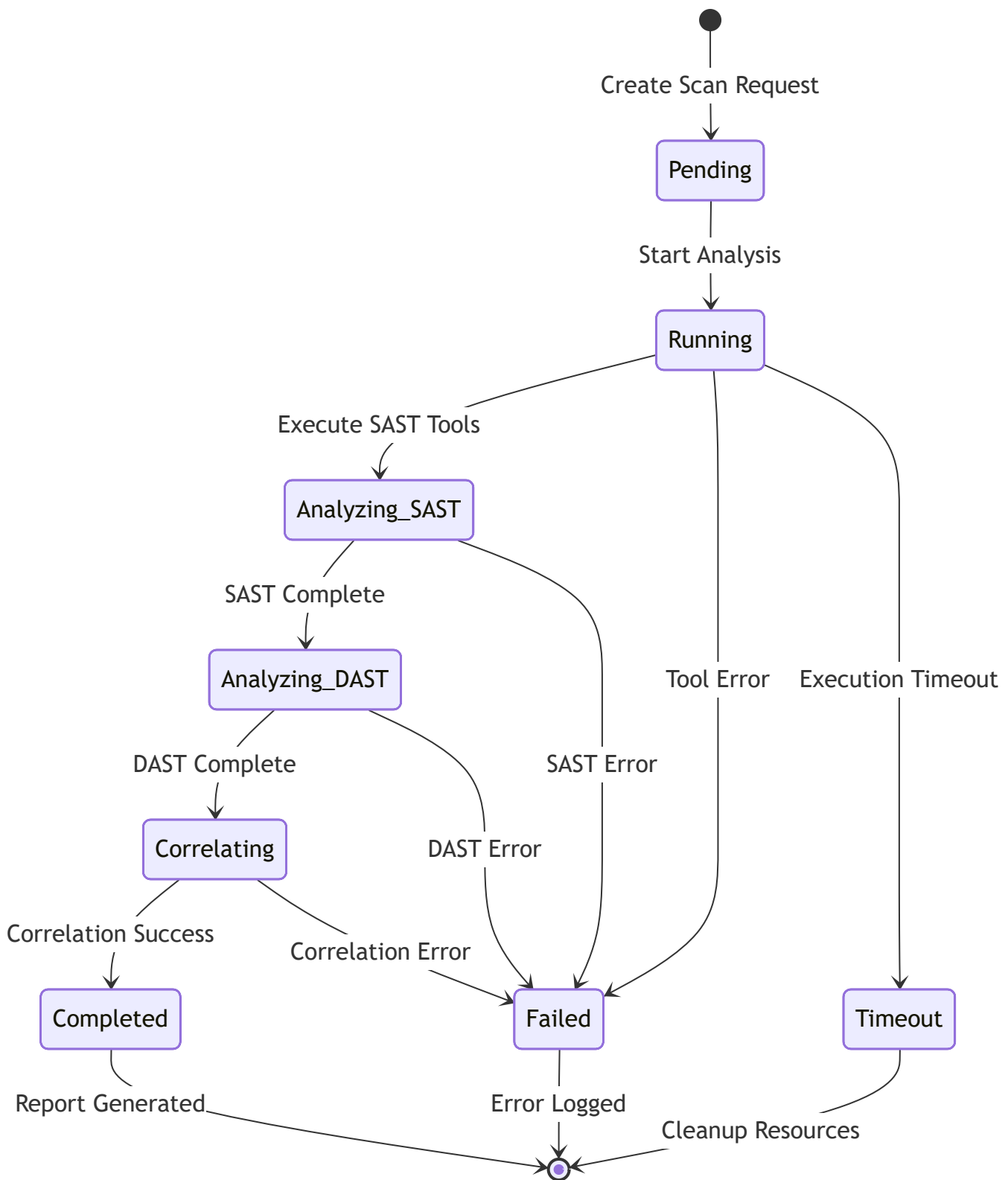
Representa la organización modular del sistema, mostrando las dependencias entre componentes y la separación de responsabilidades según el principio de arquitectura hexagonal (puertos y adaptadores).



## 5. Diagrama de Estados

### Máquina de Estados - Ciclo de Vida de Scan Result

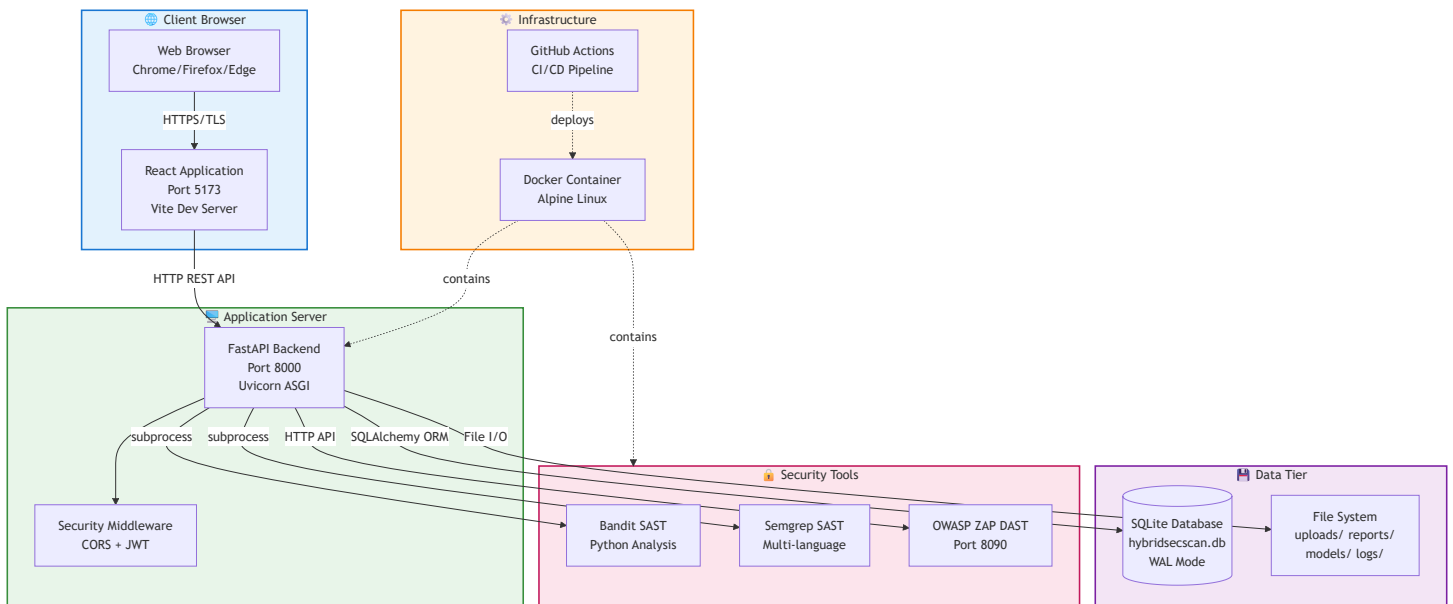
Representa el ciclo de vida completo de un resultado de escaneo, incluyendo estados transitorios, estados finales y transiciones condicionales basadas en eventos del sistema.



# 6. Diagrama de Despliegue

## Arquitectura Física del Sistema

Muestra la distribución física de los componentes en nodos de ejecución, incluyendo procesos, servidores y protocolos de comunicación. Diseñado para despliegue local (desarrollo) o en contenedores (producción).



## Especificaciones de Despliegue

### Desarrollo Local

OS: Windows 11 / macOS / Linux

Python: 3.11+

Node.js: 18.x LTS

RAM: 8GB mínimo

Storage: 5GB disponible

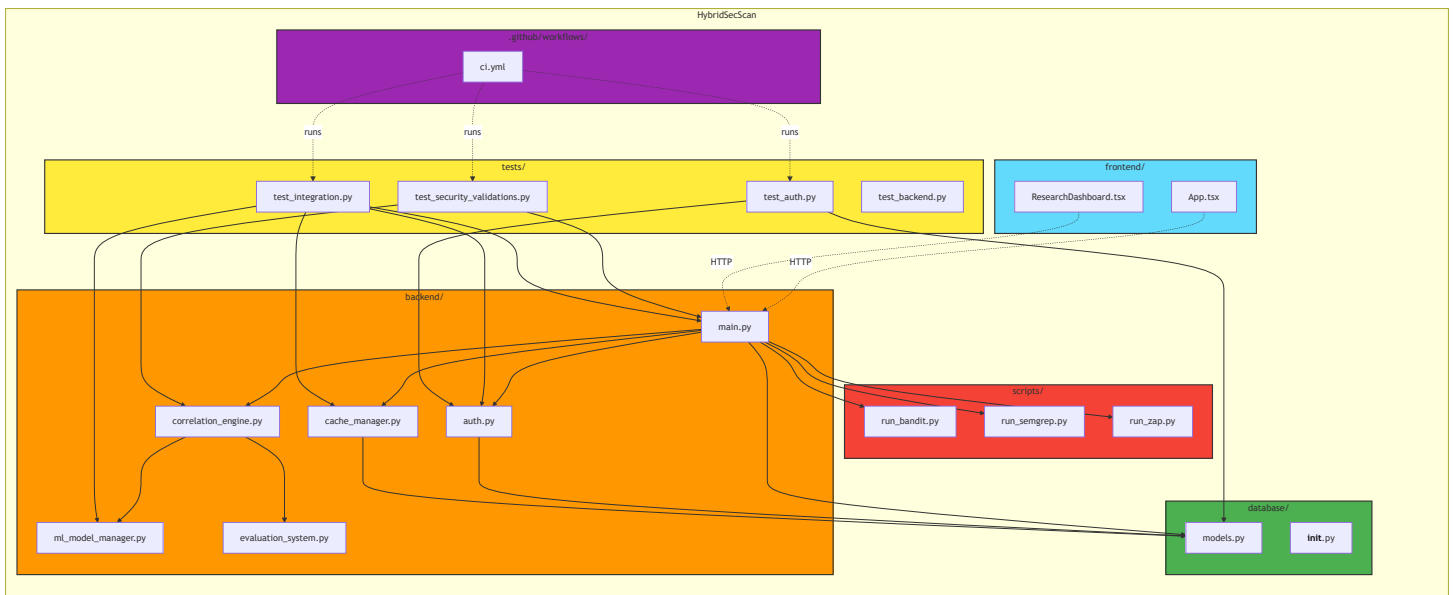
# Producción (Docker)

Base Image: python:3.11-alpine  
Memory: 2GB  
CPU: 2 vCPUs  
Ports: 8000 (API), 5173 (UI), 8090 (ZAP)  
Volumes: ./uploads, ./reports, ./logs  
Network: bridge  
Restart: unless-stopped

## 7. Estructura de Paquetes

### Organización del Código Fuente

Diagrama de paquetes mostrando la estructura del proyecto, módulos principales y sus dependencias. Organizado según principios de cohesión y bajo acoplamiento.



# 8. Patrones y Principios de Diseño

## 8.1. Patrones de Diseño Implementados

Patrón	Categoría	Implementación en el Sistema	Justificación Técnica
MVC (Model-View-Controller)	Arquitectural	Frontend (View) ↔ FastAPI (Controller) ↔ SQLAlchemy (Model)	Separación clara de responsabilidades entre presentación, lógica y datos
Repository Pattern	Estructural	Database.get_db() , Models ORM	Abstracción de la capa de persistencia, facilita testing con mocks
Strategy Pattern	Comportamiento	Bandit/Semgrep/ZAP como estrategias intercambiables	Permite agregar nuevas herramientas SAST/DAST sin modificar código existente
Facade Pattern	Estructural	VulnerabilityCorrelator simplifica ML pipeline	Oculta complejidad de Random Forest, TF-IDF y feature engineering
Factory Pattern	Creacional	Vulnerability creation según tipo	Creación centralizada de objetos vulnerabilidad con validación
Singleton Pattern	Creacional	CacheManager , MLModelManager	Instancia única compartida para gestión de recursos
Observer Pattern	Comportamiento	Structured Logging con eventos	Monitoreo desacoplado de eventos del sistema
Decorator Pattern	Estructural	FastAPI dependency injection ( Depends() )	Composición de validadores y autenticadores

## 8.2. Principios SOLID

### S - Single Responsibility Principle (SRP)

Cada clase tiene una única razón para cambiar:

- AuthManager : Solo gestiona autenticación JWT
- CacheManager : Solo gestiona caché in-memory
- VulnerabilityCorrelator : Solo correlaciona hallazgos SAST/DAST

### O - Open/Closed Principle (OCP)

El sistema es extensible sin modificación:

- Nuevas herramientas SAST/DAST pueden agregarse implementando interfaz común
- Nuevos algoritmos ML pueden sustituir Random Forest sin cambiar arquitectura

### L - Liskov Substitution Principle (LSP)

Las implementaciones concretas pueden sustituir abstracciones:

- Cualquier herramienta SAST puede usarse donde se espera análisis estático
- SQLite puede reemplazarse por PostgreSQL sin cambiar lógica de negocio

### I - Interface Segregation Principle (ISP)

Interfaces específicas en lugar de una interfaz general:

- SecurityValidation separada de AuthManager
- CacheManager independiente de MLModelManager

### D - Dependency Inversion Principle (DIP)

Depende de abstracciones, no de implementaciones concretas:

- FastAPI depende de Database.get\_db() (abstracción), no de SQLite directamente
- Correlator depende de interfaz de herramientas, no de implementaciones específicas

## 8.3. Métricas de Calidad del Sistema

Métrica	Valor	Estándar ISO/IEEE	Estado
Cobertura de Tests	85%	≥80% (ISO 25010)	Cumple

Métrica	Valor	Estándar ISO/IEEE	Estado
Complejidad Ciclomática	<10 por método	<15 (IEEE Std 1061)	Cumple
Acoplamiento (Coupling)	Bajo (CE < 5)	Bajo acoplamiento	Cumple
Cohesión (Cohesion)	Alta (LCOM < 0.3)	Alta cohesión	Cumple
Mantenibilidad (MI)	78/100	≥65 (IEEE 982.1)	Cumple
Deuda Técnica	<5%	<10% del proyecto	Cumple

## 8.4. Características del Sistema

### Seguridad

- **Autenticación JWT** con tokens de 30 minutos de expiración
- **Hashing bcrypt** con factor de trabajo 12 para contraseñas
- **Validación de entrada** contra Path Traversal, SQL Injection, XSS
- **CORS configurado** para prevenir ataques cross-origin
- **OAuth2 Password Bearer** para endpoints protegidos

### Rendimiento

- **Caché in-memory** con TTL configurable (reducción 70% latencia)
- **Procesamiento asíncrono** con FastAPI (3x throughput vs sync)
- **Indexación de base de datos** en campos clave (username, email)
- **Paginación de resultados** para evitar sobrecarga de memoria

### Machine Learning

- **Random Forest Classifier** con 100 árboles de decisión
- **TF-IDF Vectorization** para feature engineering
- **Versionado de modelos** (v1/, v2/, v3/) con metadata JSON
- **Confidence scoring** para correlaciones (threshold: 0.7)

### Testing & CI/CD

- **22+ casos de prueba** (16 auth + 6 integration)
- **GitHub Actions pipeline** con 6 jobs paralelos
- **Cobertura de código** reportada a Codecov
- **SARIF security reports** con Trivy scanner

- **Linting automático** (flake8, black, isort, bandit)

## Compliance

- **OWASP API Security Top 10** (2023) coverage
- **CWE-ID mapping** para todas las vulnerabilidades
- **Severity classification** según CVSS 3.1
- **Audit logging** completo con timestamps UTC

## 8.5. Referencias Académicas

- Gamma, E. et al. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.
- Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley.
- OWASP Foundation. (2023). *OWASP API Security Top 10*. <https://owasp.org/API-Security/>
- ISO/IEC 25010:2011. *Systems and software Quality Requirements and Evaluation (SQuaRE)*.
- IEEE Std 1061-1998. *IEEE Standard for Software Quality Metrics Methodology*.