**ApiScan**

**AI-Powered API Reliability & Governance Platform**

**Tagline:** *The CI-First Gatekeeper for Backend API Stability*

---

**1. Introduction**

In modern software systems, backend APIs act as long-living contracts between multiple consumers such as frontend applications, mobile apps, microservices, and third-party integrations. Any unintended change in an API can lead to system failures, production outages, or security vulnerabilities.

Despite this critical role, most CI/CD pipelines today focus only on **code-level correctness** and ignore **API contract stability**.

**ApiScan** is designed to solve this exact problem by acting as an **automated API governance and reliability platform** that integrates directly into CI/CD pipelines.

---

**2. Problem Statement**

Current API testing and CI tools suffer from several limitations:

1. Breaking API changes often go unnoticed until production
2. Manual test writing is slow and incomplete
3. No automated governance on API specifications
4. Security regressions are rarely tested automatically
5. Existing tools focus on execution, not prevention
6. No historical visibility of API evolution
7. AI-based testing tools lack human control

These gaps result in unstable APIs, poor developer experience, and costly production incidents.

---

**3. Proposed Solution – ApiScan**

ApiScan introduces a **governance-first approach** to backend API development.

Instead of only running tests, ApiScan:

- Enforces API contracts
- Detects breaking changes early
- Uses AI to assist test planning
- Requires human approval for AI decisions
- Blocks CI pipelines automatically when rules are violated

ApiScan ensures that **no unstable or breaking API change reaches production**.

---

**4. Objectives of the Project**

- Detect breaking changes in OpenAPI/Swagger specifications
- Enforce CI blocking rules automatically
- Reduce test creation effort using AI
- Provide auditability and historical tracking
- Ensure secure execution through sandboxing
- Introduce human-in-the-loop AI governance
- Improve backend API reliability and trust

---

**5. Key Features**

- API Spec Versioning & History
- Breaking Change Detection
- AI-Generated Test Blueprints
- Human Approval Workflow
- Sandboxed Test Execution
- Reliability Scoring System
- CI/CD Integration
- Encrypted Secret Management
- Webhook Security (HMAC)
- Full Audit Logging

---

**6. High-Level System Architecture**

ApiScan follows a **three-plane architecture**:

**6.1 Control Plane (Frontend Dashboard)**

- Built using Next.js
- Used by humans
- Displays API diffs, test plans, and reports
- Allows approval of AI-generated blueprints

**6.2 Orchestration Plane (Backend API)**

- Built using FastAPI
- Manages projects, versions, rules, and CI triggers
- Exposes REST APIs
- Delegates heavy tasks to workers

**6.3 Intelligence Plane (Workers + AI + Sandbox)**

- Uses Celery workers
- Integrates Google Gemini
- Executes tests inside Docker containers
- Fully isolated execution environment

---

**7. Technology Stack**

**Frontend**

- Next.js 16
- React 19
- TypeScript
- Tailwind CSS
- shadcn/ui
- Axios

**Backend**

- Python
- FastAPI
- SQLAlchemy
- PostgreSQL

**Workers & Async**

- Celery
- Redis

**AI**

- Google Gemini 1.5 Pro

**Sandbox**

- Docker
- Pytest

**Security**

- JWT Authentication
- AES / Fernet Encryption
- HMAC Webhook Verification

**CI/CD**

- GitHub Actions

---

**8. Database Design**

ApiScan uses PostgreSQL to maintain **immutable historical data**.

**Core Tables**

- Project
- ApiVersion
- BreakingChangeLog
- TestBlueprint
- TestRun
- SecretVault
- AuditLog

Each table is designed to ensure traceability, governance, and auditability.

---

**9. Project Entity**

The Project entity represents a backend system being governed.

Stores:

- Base API URL
- GitHub repository
- Webhook secret
- CI enforcement flags

All other entities are linked to a project.

---

**10. API Versioning**

Every uploaded OpenAPI specification:

- Is hashed (SHA-256)

- Stored immutably
- Compared against the previous version

This allows ApiScan to track API evolution over time.

---

**11. Breaking Change Detection**

ApiScan uses **DeepDiff** to compare API specs.

**Change Classification**

- **CRITICAL:** Endpoint removed, type changed
- **MAJOR:** Behavior changed
- **MINOR:** New field or endpoint added
- **INFO:** Documentation or metadata change

CRITICAL changes automatically block CI.

---

**12. AI Test Blueprint Generation**

Google Gemini analyzes:

- API spec
- API diff
- Endpoint changes

It generates **structured test blueprints**, not executable code.

These blueprints describe:

- What to test
- Which endpoints
- What scenarios (happy path, validation, security)

---

**13. Human-in-the-Loop Governance**

AI-generated blueprints:

- Are **never executed automatically**
- Must be reviewed and approved by humans
- Provide transparency and trust

This prevents unsafe AI behavior.

---

**14. Test Execution Engine**

Approved blueprints are executed as follows:

1. Docker container is created
2. Secrets injected as environment variables
3. Python test runner executes API calls
4. Results collected
5. Container destroyed

This guarantees isolation and security.

---

**15. Reliability Scoring**

Each test run produces a **Reliability Score (0–100)** based on:

- Test pass rate
- Error severity
- Response correctness

CI pipelines can enforce minimum thresholds.

---

**16. Secret Management (Vault)**

ApiScan never stores secrets in plaintext.

- Secrets encrypted at rest
- Decrypted only in memory
- Injected into Docker containers at runtime

---

**17. Webhook Security**

CI webhooks are protected using:

- HMAC SHA-256 signatures
- Project-specific secrets

Unauthenticated requests are rejected.

---

**18. CI/CD Integration**

ApiScan integrates with GitHub Actions:

- OpenAPI spec extracted during CI

- Spec uploaded to ApiScan
- CI blocked automatically on violations

---

**19. Full Project Folder Structure**

```
apiscan/
├── frontend/
│   ├── app/
│   ├── components/
│   ├── hooks/
│   ├── lib/
│   └── package.json
├── backend/
│   ├── app/
│   │   ├── models/
│   │   ├── routers/
│   │   ├── services/
│   │   ├── security.py
│   │   └── main.py
│   ├── alembic/
│   └── requirements.txt
├── worker/
│   ├── app/
│   │   ├── tasks/
│   │   └── ai/
│   └── requirements.txt
├── runner/
│   ├── Dockerfile
│   └── entrypoint.py
├── ci/
│   └── github/
```

├── infra/

└── docs/

---

**20. Future Scope**

- Role-Based Access Control (RBAC)
- Multi-environment policies
- Advanced security fuzzing
- SSO integration
- Kubernetes-native execution
- API drift visualization

---

**21. Conclusion**

ApiScan is not just an API testing tool.
It is a **complete API reliability and governance platform**.

By combining CI enforcement, AI assistance, security, and human oversight, ApiScan ensures backend APIs remain stable, secure, and production-ready.

📁 **ApiScan – Full Folder & File Structure**

apiscan/

|

├── frontend/ # Control Plane (UI / Dashboard)

| ├── app/ # Next.js App Router

| | ├── (auth)/ # Auth routes (login/signup)

| | | ├── login/

| | | └── register/

| | ├── dashboard/

| | | ├── page.tsx # Main dashboard

| | | ├── projects/

| | | | ├── page.tsx

| | | | └── [projectId]/

| | | | ├── page.tsx

```
|   |   |   |       ├── versions/
|   |   |   |       ├── blueprints/
|   |   |   |       └── runs/
|   |   |   └── settings/
|   |   ├── layout.tsx
|   |   └── globals.css
|   |
|   ├── components/              # Reusable UI components
|   |   ├── ui/                  # shadcn/ui components
|   |   ├── charts/
|   |   ├── tables/
|   |   ├── modals/
|   |   └── loaders/
|   |
|   ├── hooks/                   # Custom React hooks
|   |   ├── useAuth.ts
|   |   ├── useProjects.ts
|   |   └── useApiVersions.ts
|   |
|   ├── lib/                     # Frontend utilities
|   |   ├── api.ts               # Axios instance
|   |   ├── auth.ts
|   |   └── constants.ts
|   |
|   ├── types/                   # TypeScript types
|   |   ├── project.ts
|   |   ├── apiVersion.ts
|   |   └── testRun.ts
|   |
|   ├── package.json
```

```
|   ├── tsconfig.json
|   └── tailwind.config.ts
|
├── backend/                    # Orchestration Plane (API)
|   ├── app/
|   |   ├── main.py               # FastAPI entry point
|   |   |
|   |   ├── core/                 # Core configs
|   |   |   ├── config.py
|   |   |   ├── logging.py
|   |   |   └── security_config.py
|   |   |
|   |   ├── database.py           # DB connection/session
|   |   |
|   |   ├── models/               # SQLAlchemy models
|   |   |   ├── project.py
|   |   |   ├── api_version.py
|   |   |   ├── breaking_change.py
|   |   |   ├── test_blueprint.py
|   |   |   ├── test_run.py
|   |   |   ├── secret_vault.py
|   |   |   └── audit_log.py
|   |   |
|   |   ├── schemas/              # Pydantic schemas
|   |   |   ├── auth.py
|   |   |   ├── project.py
|   |   |   ├── specs.py
|   |   |   ├── test_blueprint.py
|   |   |   ├── test_run.py
|   |   |   └── secret_vault.py
```

```
|   |   |
|   |   ├── routers/              # API routes
|   |   |   ├── auth.py
|   |   |   ├── projects.py
|   |   |   ├── specs.py
|   |   |   ├── test_blueprints.py
|   |   |   ├── test_runs.py
|   |   |   ├── secrets.py
|   |   |   └── webhooks.py
|   |   |
|   |   ├── services/             # Business logic
|   |   |   ├── diff_service.py       # Breaking change detection
|   |   |   ├── blueprint_service.py  # AI blueprint orchestration
|   |   |   └── scoring_service.py
|   |   |
|   |   ├── security.py           # JWT, encryption, auth helpers
|   |   └── utils/
|   |       ├── hashing.py
|   |       └── time.py
|   |
|   ├── alembic/                  # Database migrations
|   |   ├── versions/
|   |   └── env.py
|   |
|   ├── requirements.txt
|   └── Dockerfile
|
├── worker/                     # Intelligence Plane (Async Workers)
|   ├── app/
|   |   ├── main.py               # Celery app
```

```
|   |   |
|   |   ├── tasks/                # Background tasks
|   |   |   ├── generate_blueprint.py
|   |   |   ├── execute_test_run.py
|   |   |   └── cleanup_runs.py
|   |   |
|   |   ├── ai/                   # AI integrations
|   |   |   ├── gemini_client.py
|   |   |   └── prompt_templates.py
|   |   |
|   |   └── utils/
|   |       ├── docker.py
|   |       └── retry.py
|   |
|   ├── requirements.txt
|   └── Dockerfile
|
├── runner/                       # Sandboxed Execution Engine
|   ├── Dockerfile
|   ├── entrypoint.py             # Executes tests inside container
|   ├── runner.py                 # HTTP test executor
|   └── requirements.txt
|
├── ci/                           # CI/CD integration
|   └── github/
|       ├── action.yml
|       └── upload_spec.sh
|
├── infra/                        # Infrastructure configs
|   ├── docker-compose.yml
```

```
|   ├── redis.yml
|   └── postgres.yml
|
├── docs/                       # Documentation
|   ├── architecture.md
|   ├── api_contracts.md
|   └── diagrams/
|
├── .env.example
├── .gitignore
├── README.md
└── LICENSE
```