

DevGuard

Spec final completo para o agente de desenvolvimento

v0.1.0 · Agent Security · CLI Python · PyPI · SQLite memory

INSTRUÇÃO CRÍTICA PARA O AGENTE: Antes de implementar `auth.py`, leia obrigatoriamente os dois projetos descritos na **Seção 2**. Não invente o contrato da API — leia o código real.

Índice

1. Estrutura de arquivos completa
2. PASSO 0 — Leitura obrigatória antes de codificar
3. `pyproject.toml`
4. `devguard/__init__.py`
5. `devguard/config.py` — detecção de API keys
6. `devguard/auth.py` — validação de licença
7. `devguard/tools.py` — schema + implementação das 5 tools
8. `devguard/prompts.py` — system prompt do Agent Security
9. `devguard/adapter.py` — abstração multi-model
10. `devguard/memory.py` — SQLite entre análises (NOVO)
11. `devguard/agent.py` — loop agentic com memória
12. `devguard/cli.py` — interface de linha de comando
13. `.env.example` e `README.md`
14. Ordem de implementação passo a passo
15. Como publicar no PyPI e testar

```
documents/guardion-ai/backend/agent_security/
■■■ devguard/
■   ■■■ __init__.py           # versão do pacote
■   ■■■ cli.py               # entry point: devguard security / version
■   ■■■ agent.py             # loop agentic com injeção de memória
■   ■■■ adapter.py           # abstração Claude / GPT-4o / Groq
■   ■■■ tools.py             # schema + execução das 5 tools
■   ■■■ prompts.py           # system prompts especializados
■   ■■■ memory.py            # SQLite local – memória entre análises ← NOVO
■   ■■■ auth.py              # validação de licença
■   ■■■ config.py            # detecção de API keys do ambiente
■■■ tests/
■   ■■■ test_tools.py
■   ■■■ test_memory.py
■   ■■■ test_adapter.py
■   ■■■ test_auth.py
■■■ pyproject.toml
■■■ .env.example
■■■ README.md

# Gerado em runtime na máquina do usuário (não vai pro git):
.devguard/
■■■ devguard.db              # SQLite com histórico de findings por projeto
```

O backend em `documents/guardion-ai/backend` já possui validação de API key implementada e funcionando. O DevGuard vai ser criado DENTRO desse mesmo backend, na pasta `agent_security/`. A lógica de validação já está lá — só reutilize. Não invente nada, não crie nada novo fora de `agent_security/`.

O que o agente deve fazer

O backend `documents/guardion-ai/backend` já valida API keys. O agente deve ler o código existente nesse backend, entender como a validação funciona, e reutilizar exatamente isso no `auth.py` do DevGuard. Nenhum arquivo fora de `agent_security/` deve ser criado ou modificado.

- Abrir `documents/guardion-ai/backend/` e localizar onde a validação de API key está implementada
- Ler o código de validação existente — pode ser `middleware`, `guard`, `service` ou `decorator`
- Entender o contrato: como a key chega, como é validada, o que é retornado
- Implementar `auth.py` dentro de `agent_security/` reutilizando essa lógica
- Não criar nada, não modificar nada fora de `agent_security/`

Escopo do agente — `agent_security/` APENAS

Cria e modifica arquivos SOMENTE dentro de `agent_security/`

Lê o restante do backend apenas para entender a validação existente

Nunca escreve ou altera arquivos fora de `agent_security/`

```
[project]
name = "devguard"
version = "0.1.0"
description = "Autonomous AI security agent for your codebase"
readme = "README.md"
requires-python = ">=3.10"
license = {text = "MIT"}
keywords = ["security", "ai", "agent", "owasp", "sast", "pentest"]
dependencies = [
    "anthropic>=0.25",
    "openai>=1.0",
    "click>=8.0",
    "rich>=13.0",
    "httpx>=0.27",
]

# Este bloco transforma o pacote num comando CLI.
# Após 'pip install devguard', o comando 'devguard' fica disponível no terminal.
[project.scripts]
devguard = "devguard.cli:main"

[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"
```

4

devguard/__init__.py

Python

```
__version__ = "0.1.0"
```

Lê as variáveis de ambiente e detecta qual provider de LLM usar. Ordem de prioridade: Anthropic > OpenAI > Groq. Falha com mensagem clara se nenhuma key for encontrada.

```

import os
from dataclasses import dataclass

VALIDATE_URL_DEFAULT = "https://api.guardion.ai/api/validate-key"
# URL do endpoint de validação – confirmar no backend após Passo 0

@dataclass
class Config:
    devguard_key: str
    provider: str          # "anthropic" | "openai" | "groq"
    llm_key: str
    model: str
    validate_url: str

def load_config(model_override: str | None = None) -> Config:
    devguard_key = os.getenv("DEVGUARD_API_KEY")
    if not devguard_key:
        raise SystemExit(
            "DEVGUARD_API_KEY nao encontrada.\n"
            "Configure: export DEVGUARD_API_KEY=dg...\n"
            "Obtenha em: https://guardion.ai/devguard"
        )

    if key := os.getenv("ANTHROPIC_API_KEY"):
        provider, llm_key = "anthropic", key
        default_model = "claude-sonnet-4-20250514"
    elif key := os.getenv("OPENAI_API_KEY"):
        provider, llm_key = "openai", key
        default_model = "gpt-4o"
    elif key := os.getenv("GROQ_API_KEY"):
        provider, llm_key = "groq", key
        default_model = "llama-3.1-70b-versatile"
    else:
        raise SystemExit(
            "Nenhuma API key de LLM encontrada.\n"
            "Configure uma das variaveis:\n"
            "  ANTHROPIC_API_KEY  -> Claude Sonnet (recomendado)\n"
            "  OPENAI_API_KEY     -> GPT-4o\n"
            "  GROQ_API_KEY        -> Llama 3.1 70B (mais barato)"
        )

    return Config(
        devguard_key=devguard_key,
        provider=provider,
        llm_key=llm_key,
        model=model_override or default_model,
        validate_url=os.getenv("DEVGUARD_VALIDATE_URL", VALIDATE_URL_DEFAULT)
    )

```

6

devguard/auth.py — validação de licença

Implementar APÓS executar o Passo 0

Implementar SOMENTE após executar o Passo 0. O contrato abaixo tem placeholders marcados com TODO. O agente substitui cada TODO pelo valor real encontrado no backend existente.

Este módulo valida a `DEVGUARD_API_KEY` contra o backend do `guardion-ai` via HTTP. A validação já está implementada no backend — o agente só precisa ler como funciona e reutilizar no DevGuard.


```

import httpx
from .config import Config

class AuthError(Exception):
    pass

def validate_license(config: Config) -> dict:
    # TODO: apos executar o Passo 0, confirmar e substituir:
    # - URL real do endpoint de validacao
    # - Headers corretos (Authorization, X-API-Key, etc)
    # - Body real da request (pode ser vazio)
    # - Campo da response que indica sucesso/falha

    url = config.validate_url # TODO: confirmar URL real

    headers = {
        "Authorization": f"Bearer {config.devguard_key}",
        "Content-Type": "application/json",
    }

    body = {"service": "devguard"} # TODO: confirmar body real

    try:
        resp = httpx.post(url, headers=headers, json=body, timeout=10.0)
    except httpx.ConnectError:
        raise AuthError(
            "Nao foi possivel conectar ao servidor de licencas.\n"
            "Verifique sua conexao com a internet."
        )
    except httpx.TimeoutException:
        raise AuthError("Timeout ao validar licenca. Tente novamente.")

    if resp.status_code == 200:
        data = resp.json()
        if data.get("valid"): # TODO: confirmar campo real
            return data
        reason = data.get("reason", "unknown")
        raise AuthError(f"Licenca invalida: {reason}")

    if resp.status_code in (401, 403):
        raise AuthError(
            "DEVGUARD_API_KEY invalida ou expirada.\n"
            "Renove em: https://guardion.ai/devguard"
        )

    raise AuthError(f"Erro inesperado: HTTP {resp.status_code}")

```



```

import subprocess, os, json, urllib.request, urllib.error
from pathlib import Path

IGNORE_DIRS = {
    "node_modules", ".git", "__pycache__", "venv", ".venv",
    "env", "dist", "build", ".next", ".nuxt", "coverage",
    ".pytest_cache", ".devguard"
}

TOOLS_SCHEMA = [
    {
        "name": "run_command",
        "description": (
            "Executa um comando shell na maquina do usuario e retorna stdout+stderr. "
            "Use para rodar ferramentas de seguranca (nmap, trivy, semgrep, gitleaks, "
            "docker), inspecionar o projeto, subir servicos. "
            "Verifique disponibilidade da ferramenta antes de usar. "
            "Prefira flags --json quando disponiveis. Timeout default 120s."
        ),
        "input_schema": {
            "type": "object",
            "properties": {
                "command": {"type": "string"},
                "cwd": {"type": "string"},
                "timeout": {"type": "integer"}
            },
            "required": ["command"]
        },
    },
    {
        "name": "read_file",
        "description": "Le o conteudo de um arquivo do projeto.",
        "input_schema": {
            "type": "object",
            "properties": {"path": {"type": "string"}},
            "required": ["path"]
        },
    },
    {
        "name": "list_dir",
        "description": (
            "Lista estrutura de arquivos ate 3 niveis. "
            "Ignora node_modules, .git, __pycache__, venv, .devguard. "
            "Use como PRIMEIRO passo obrigatorio em toda analise."
        ),
        "input_schema": {
            "type": "object",
            "properties": {"path": {"type": "string"}},
            "required": ["path"]
        },
    },
]

```

```

    },
    {
        "name": "http_request",
        "description": (
            "Faz requisicao HTTP para testar endpoints. "
            "Use para verificar headers de seguranca, autenticacao, CORS, "
            "endpoints expostos, e se o servico esta publico."
        ),
        "input_schema": {
            "type": "object",
            "properties": {
                "url": {"type": "string"},
                "method": {"type": "string",
                           "enum": ["GET", "POST", "PUT", "DELETE", "HEAD", "OPTIONS"]},
                "headers": {"type": "object"},
                "body": {"type": "string"}
            },
            "required": ["url"]
        }
    },
    {
        "name": "write_file",
        "description": (
            "Escreve um arquivo. Use APENAS para salvar o relatorio final "
            "em devguard-report.md. NUNCA modifique codigo do usuario."
        ),
        "input_schema": {
            "type": "object",
            "properties": {
                "path": {"type": "string"},
                "content": {"type": "string"}
            },
            "required": ["path", "content"]
        }
    }
]

def to_openai_schema(tools: list) -> list:
    return [{"type": "function", "function": {
        "name": t["name"],
        "description": t["description"],
        "parameters": t["input_schema"]
    }} for t in tools]

def execute_tool(name: str, inputs: dict, project_path: str) -> str:
    try:
        match name:
            case "run_command": return _run(inputs, project_path)
            case "read_file": return _read(inputs, project_path)
            case "list_dir": return _ls(inputs, project_path)

```

```

        case "http_request": return _http(inputs)
        case "write_file":   return _write(inputs, project_path)
        case _:              return f"Tool desconhecida: {name}"
except Exception as e:
    return f"ERRO em {name}: {type(e).__name__}: {e}"

def _run(inputs, project_path):
    cwd = inputs.get("cwd") or project_path
    result = subprocess.run(
        inputs["command"], shell=True, cwd=cwd,
        capture_output=True, text=True,
        timeout=inputs.get("timeout", 120)
    )
    out = result.stdout + result.stderr
    if len(out) > 12000:
        out = out[:6000] + "\n\n[... truncado ...]\n\n" + out[-3000:]
    return out or "(sem output)"

def _read(inputs, project_path):
    path = inputs["path"]
    if not os.path.isabs(path):
        path = os.path.join(project_path, path)
    with open(path, encoding="utf-8", errors="replace") as f:
        content = f.read()
    if len(content) > 15000:
        content = content[:15000] + "\n\n[... truncado ...]"
    return content

def _ls(inputs, project_path):
    base = inputs.get("path") or project_path
    if not os.path.isabs(base):
        base = os.path.join(project_path, base)
    lines = []
    for root, dirs, files in os.walk(base):
        dirs[:] = sorted(d for d in dirs if d not in IGNORE_DIRS)
        depth = len(Path(root).relative_to(base).parts)
        if depth > 3: continue
        lines.append("  " * depth + os.path.basename(root) + "/")
        for f in sorted(files):
            lines.append("  " * depth + "  " + f)
    return "\n".join(lines)

def _http(inputs):
    url = inputs["url"]
    method = inputs.get("method", "GET").upper()
    headers = inputs.get("headers") or {}
    body = inputs.get("body", "").encode() if inputs.get("body") else None
    req = urllib.request.Request(url, data=body, headers=headers, method=method)
    try:
        with urllib.request.urlopen(req, timeout=15) as r:

```

```
        return json.dumps({
            "status": r.status,
            "headers": dict(r.headers),
            "body": r.read(4000).decode("utf-8", errors="replace")
        }, indent=2)
    except urllib.error.HTTPError as e:
        return f"HTTP {e.code}: {e.reason}"
    except Exception as e:
        return f"Erro: {e}"

def _write(inputs, project_path):
    path = inputs["path"]
    if not os.path.isabs(path):
        path = os.path.join(project_path, path)
    Path(path).parent.mkdir(parents=True, exist_ok=True)
    with open(path, "w", encoding="utf-8") as f:
        f.write(inputs["content"])
    return f"Salvo em: {path}"
```



```

SECURITY_SYSTEM_PROMPT = ""

```

Voce e o Agent Security do DevGuard – agente autonomo de seguranca ofensiva e defensiva. Aja como pentester senior: encontra vulnerabilidades, entende o contexto, entrega remediações prontas para aplicar.

```

## Regras de comportamento

REGRA 1: Seja autonomo. Nunca peca confirmacao antes de executar ferramentas.
        Confirme APENAS antes de modificar arquivos do usuario (write_file em codigo).
REGRA 2: Verifique disponibilidade. Antes de usar nmap/semgrep/gitleaks:
        run_command('which <ferramenta> 2>/dev/null || echo NOT_FOUND')
        Se NOT_FOUND, tente via Docker. Se Docker tambem nao tiver, documente.
REGRA 3: Adapte-se a stack. Node.js -> npm audit. Python -> pip-audit.
        Dockerfile -> trivy image. Nao execute ferramentas irrelevantes.
REGRA 4: Extraia findings imediatamente ao receber output de ferramenta.
REGRA 5: Correlacione. Secret do gitleaks + porta aberta do nmap = risco maior.

## Fase 1 – Reconhecimento (SEMPRE executar primeiro)

1. list_dir(project_path) -> entenda stack, linguagem, framework, Docker
2. Leia arquivos de dependencia: package.json / requirements.txt / pom.xml / go.mod
3. run_command('find . -name ".env*" -o -name "*.pem" -o -name "*.key" 2>/dev/null')
4. run_command('git log --oneline -20') se .git existir

## Fase 2 – SAST (analise estatica, sem rede)

Execute todos os disponiveis:
- Secrets: gitleaks detect --source=. --report-format=json --no-banner
  Fallback: grep -rn 'password|secret|api_key|token' . (excluindo .git)
- SAST: semgrep --config=auto --json --quiet .
  Fallback: docker run --rm -v $(pwd):/src returntocorp/semgrep semgrep ...
- Deps Python: pip-audit --format=json
- Deps Node: npm audit --json
- Filesystem: docker run --rm -v $(pwd):/project aquasec/trivy fs --format json /project
- Leia manualmente: arquivos de auth, configs, controllers, conexoes DB

## Fase 3 – Analise dinamica (servico rodando)

1. Verifique se esta publico: busque URLs no README/.env.example, teste com http_request
2. Se nao publico e tiver docker-compose.yml:
  run_command('docker compose up -d --wait', timeout=120)
  run_command('docker compose ps')
3. Com servico up (local ou publico):
  - nmap -sV --open -p 1-10000 localhost
  - http_request(url) -> analise todos os headers de seguranca
  - docker run owasp/zap2docker-stable zap-baseline.py -t {url} (timeout=300)
  - docker run projectdiscovery/nuclei -u {url} -t cves/ (timeout=300)
4. Cleanup: docker stop/rm do container que voce subiu

## Fase 4 – Autenticacao e autorizacao

```


Analise o código para:

- JWT: algoritmo (rejeitar alg:none), expiracao, secret em env var
- Cookies: HttpOnly, Secure, SameSite=Strict/Lax
- OAuth: state parameter, PKCE, redirect_uri whitelist
- RBAC: endpoints sem autenticação, IDOR

Fase 5 – Relatório final

Gere o relatório neste formato e salve em {project_path}/devguard-report.md:

DevGuard Security Report

Projeto: {nome} | **Data:** {data} | **Stack:** {stack}

Sumário executivo

{3-5 frases diretas. Ex: 'O projeto tem 2 vulnerabilidades críticas...'} }

Findings críticos – CVSS >= 7.0

[CRÍTICO] {titulo}

CVSS: {score} | **CWE:** {num} | **Ferramenta:** {qual}

Localizacao: {arquivo:linha ou endpoint}

Descricao: {o que e, por que e grave}

PoC: `{comando ou payload que demonstra}`

Remediacao: {codigo corrigido pronto para copiar}

Findings médios – CVSS 4.0-6.9 (mesmo formato)

Findings baixos / informativos (lista: item | local | recomendacao)

Headers de segurança (tabela: header | status | atual | recomendado)

Dependências vulneráveis (tabela: pacote | versao | CVE | fix)

STRIDE matrix (tabela por componente)

Verificações puladas (ferramenta | motivo)

Após salvar o relatório: informe o usuário e pare. Não faça mais tool calls.

"""

AGENT_PROMPTS = {

 "security": SECURITY_SYSTEM_PROMPT,

}


```

from dataclasses import dataclass
import json
from .config import Config
from .tools import TOOLS_SCHEMA, to_openai_schema

@dataclass
class ToolCall:
    id: str
    name: str
    inputs: dict

@dataclass
class ModelResponse:
    tool_calls: list
    text: str | None
    done: bool
    _raw: object = None

class ModelAdapter:
    def __init__(self, config: Config, system_prompt: str):
        self.config = config
        self.system_prompt = system_prompt
        if config.provider == "anthropic":
            import anthropic
            self._client = anthropic.Anthropic(api_key=config.llm_key)
        else:
            from openai import OpenAI
            base = "https://api.groq.com/openai/v1" if config.provider == "groq" else None
            self._client = OpenAI(api_key=config.llm_key, base_url=base)

    def call(self, messages: list) -> ModelResponse:
        if self.config.provider == "anthropic":
            return self._anthropic(messages)
        return self._openai(messages)

    def _anthropic(self, messages) -> ModelResponse:
        raw = self._client.messages.create(
            model=self.config.model, max_tokens=8096,
            system=self.system_prompt,
            tools=TOOLS_SCHEMA, messages=messages
        )
        tcs, text = [], None
        for b in raw.content:
            if b.type == "tool_use":
                tcs.append(ToolCall(id=b.id, name=b.name, inputs=b.input))
            elif b.type == "text":
                text = b.text
        return ModelResponse(tcs, text, raw.stop_reason == "end_turn", raw)

    def _openai(self, messages) -> ModelResponse:

```

```

sys = [{"role": "system", "content": self.system_prompt}]
raw = self._client.chat.completions.create(
    model=self.config.model, max_tokens=8096,
    tools=to_openai_schema(TOOLS_SCHEMA),
    messages=sys + messages
)
ch = raw.choices[0]
tcs = []
if ch.message.tool_calls:
    for tc in ch.message.tool_calls:
        tcs.append(ToolCall(
            id=tc.id, name=tc.function.name,
            inputs=json.loads(tc.function.arguments)
        ))
return ModelResponse(tcs, ch.message.content, ch.finish_reason == "stop", raw)

def assistant_msg(self, resp: ModelResponse) -> dict:
    if self.config.provider == "anthropic":
        return {"role": "assistant", "content": resp._raw.content}
    return {"role": "assistant", "content": resp._raw.choices[0].message}

def tool_results_msg(self, tcs: list, results: list) -> list:
    if self.config.provider == "anthropic":
        return [{"role": "user", "content": [
            {"type": "tool_result", "tool_use_id": tc.id, "content": r}
            for tc, r in zip(tcs, results)
        ]}]
    return [
        {"role": "tool", "tool_call_id": tc.id, "content": r}
        for tc, r in zip(tcs, results)
    ]

```

Persiste findings entre runs no arquivo `.devguard/devguard.db` na raiz do projeto. Usa `sqlite3` da `stdlib` do Python — zero dependências externas. O agent runner lê os findings abertos do run anterior e injeta como contexto no system prompt.

Como funciona a injeção de contexto

1. Antes de cada análise, `memory.load_previous_context(project_path)` lê o SQLite
2. Retorna texto formatado com os findings abertos do último run
3. `agent.py` injeta esse texto no system prompt antes de chamar o LLM
4. O agente naturalmente compara e produz relatório diferencial
5. Ao final, `memory.save_run()` persiste os novos findings no SQLite
6. Findings que sumiram do scan são marcados como 'fixed' automaticamente

```

import sqlite3
import uuid
import hashlib
import json
from datetime import datetime, timezone
from pathlib import Path

def _db_path(project_path: str) -> str:
    """Retorna o path do SQLite. Fica em .devguard/ dentro do projeto."""
    db_dir = Path(project_path) / ".devguard"
    db_dir.mkdir(exist_ok=True)
    # Garante que .devguard/devguard.db nao vai pro git
    gitignore = db_dir / ".gitignore"
    if not gitignore.exists():
        gitignore.write_text("*\n")
    return str(db_dir / "devguard.db")

def _connect(project_path: str) -> sqlite3.Connection:
    conn = sqlite3.connect(_db_path(project_path))
    conn.row_factory = sqlite3.Row
    return conn

def init_db(project_path: str):
    """Cria as tabelas se nao existirem. Idempotente."""
    with _connect(project_path) as conn:
        conn.executescript("""
            CREATE TABLE IF NOT EXISTS runs (
                id            TEXT PRIMARY KEY,
                project       TEXT NOT NULL,
                ran_at        TEXT NOT NULL,
                model         TEXT,
                stack         TEXT
            );
            CREATE TABLE IF NOT EXISTS findings (
                id            TEXT PRIMARY KEY,
                run_id        TEXT NOT NULL REFERENCES runs(id),
                title         TEXT NOT NULL,
                severity      TEXT,
                cvss          REAL,
                cwe           TEXT,
                file_path     TEXT,
                line_number   INTEGER,
                tool          TEXT,
                status        TEXT DEFAULT 'open',
                first_seen    TEXT NOT NULL,
                last_seen     TEXT NOT NULL
            );
        """)

def make_run_id() -> str:

```

```

return str(uuid.uuid4())

def finding_hash(title: str, file_path: str, line_number: int | None) -> str:
    """
    ID determinístico por finding.
    Mesmo finding em dois runs diferentes = mesmo ID.
    Permite detectar se foi corrigido entre runs.
    """
    key = f"{title}|{file_path or ''}|{line_number or 0}"
    return hashlib.sha256(key.encode()).hexdigest()[:16]

def save_run(
    project_path: str,
    run_id: str,
    model: str,
    stack: str,
    findings: list[dict]
):
    """
    Persiste um run e seus findings.

    findings: lista de dicts com campos:
        title, severity, cvss, cwe, file_path,
        line_number, tool
    """
    init_db(project_path)
    now = datetime.now(timezone.utc).isoformat()

    with _connect(project_path) as conn:
        conn.execute(
            "INSERT INTO runs (id, project, ran_at, model, stack) VALUES (?, ?, ?, ?, ?)",
            (run_id, project_path, now, model, stack)
        )
        for f in findings:
            fid = finding_hash(f["title"], f.get("file_path"), f.get("line_number"))
            # Verifica se finding ja existia (para preservar first_seen)
            existing = conn.execute(
                "SELECT first_seen FROM findings WHERE id = ?", (fid,)
            ).fetchone()
            first_seen = existing["first_seen"] if existing else now
            # Upsert: insere ou atualiza
            conn.execute("""
                INSERT INTO findings
                (id, run_id, title, severity, cvss, cwe,
                 file_path, line_number, tool, status, first_seen, last_seen)
                VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
                ON CONFLICT(id) DO UPDATE SET
                run_id=excluded.run_id, status='open',
                last_seen=excluded.last_seen
            """, (

```

```

        fid, run_id, f["title"], f.get("severity"), f.get("cvss"),
        f.get("cwe"), f.get("file_path"), f.get("line_number"),
        f.get("tool"), "open", first_seen, now
    ))

def mark_fixed_since(project_path: str, current_run_id: str):
    """
    Marca como 'fixed' todos os findings que nao apareceram no run atual.
    Chame APOS save_run() do run atual.
    """
    with _connect(project_path) as conn:
        conn.execute("""
            UPDATE findings SET status = 'fixed'
            WHERE status = 'open' AND run_id != ?
            """, (current_run_id,))

def load_previous_context(project_path: str) -> str:
    """
    Retorna texto formatado com os findings abertos do ultimo run.
    Retorna string vazia se for a primeira analise.
    Este texto e injetado no system prompt pelo agent.py.
    """
    init_db(project_path)
    with _connect(project_path) as conn:
        last_run = conn.execute(
            "SELECT id, ran_at FROM runs ORDER BY ran_at DESC LIMIT 1"
        ).fetchone()
        if not last_run:
            return "" # primeira analise

        open_findings = conn.execute("""
            SELECT title, severity, file_path, line_number,
                   tool, first_seen
            FROM findings
            WHERE status = 'open'
            ORDER BY cvss DESC NULLS LAST
            """).fetchall()

        if not open_findings:
            return f"""
## Contexto: analise anterior ({last_run['ran_at'][:10]})
Nenhum finding aberto da analise anterior. Excelente!
Verifique se novos problemas foram introduzidos desde entao.
"""

        lines = [
            f"## Contexto: analise anterior ({last_run['ran_at'][:10]})",
            f"Os findings abaixo estavam abertos na ultima analise.",
            f"Verifique se cada um foi corrigido ou ainda existe:",
            "",

```



```

    ]
    for f in open_findings:
        loc = f[f['file_path']]:{f['line_number']} if f['file_path'] else "desconhecido"
        days = (datetime.now(timezone.utc) -
                 datetime.fromisoformat(f['first_seen'])).days
        lines.append(
            f"- [{f['severity']} or 'INFO'] {f['title']} | {loc} | "
            f"aberto ha {days} dia(s) | ferramenta: {f['tool']} or 'manual'"
        )
        lines.append("")
    return "\n".join(lines)

def get_stats(project_path: str) -> dict:
    """Retorna estatísticas para exibir no início da análise."""
    init_db(project_path)
    with _connect(project_path) as conn:
        total_runs = conn.execute("SELECT COUNT(*) FROM runs").fetchone()[0]
        open_critical = conn.execute(
            "SELECT COUNT(*) FROM findings WHERE status='open' AND severity='critical'"
        ).fetchone()[0]
        open_high = conn.execute(
            "SELECT COUNT(*) FROM findings WHERE status='open' AND severity='high'"
        ).fetchone()[0]
        fixed_total = conn.execute(
            "SELECT COUNT(*) FROM findings WHERE status='fixed'"
        ).fetchone()[0]
    return {
        "total_runs": total_runs,
        "open_critical": open_critical,
        "open_high": open_high,
        "fixed_total": fixed_total,
    }

```



```

import uuid
from rich.console import Console
from rich.live import Live
from rich.text import Text
from rich.markdown import Markdown
from .adapter import ModelAdapter
from .tools import execute_tool
from .prompts import AGENT_PROMPTS, SECURITY_SYSTEM_PROMPT
from .memory import (
    load_previous_context, save_run,
    mark_fixed_since, get_stats, make_run_id
)
from .config import Config

console = Console()

def run_agent(
    agent: str,
    project_path: str,
    config: Config,
    save_report: bool = True
):
    run_id = make_run_id()

    # Exibe estatísticas do historico antes de comecar
    stats = get_stats(project_path)
    if stats["total_runs"] > 0:
        console.print(f"[dim]Historico: {stats['total_runs']} analises | "
                      f"{stats['open_critical']} criticos abertos | "
                      f"{stats['fixed_total']} corrigidos[/dim]")

    # Carrega contexto anterior e injeta no system prompt
    prev_ctx = load_previous_context(project_path)
    base_prompt = AGENT_PROMPTS[agent]
    system_prompt = base_prompt + ("\n\n" + prev_ctx if prev_ctx else "")

    adapter = ModelAdapter(config, system_prompt)

    messages = [{
        "role": "user",
        "content": (
            f"Analise o projeto em: {project_path}\n"
            "Execute todas as fases de forma autonoma e completa.\n"
            f"Salve o relatorio em: {project_path}/devguard-report.md"
        )
    }]

    console.print(f"\n[bold]DevGuard[/bold] · Agent {agent.title()}")
    console.print(f"[dim]Projeto: {project_path}[/dim]")
    console.print(f"[dim]Modelo: {config.provider}/{config.model}[/dim]\n")

```

```

findings_extracted = [] # coletados durante o run para persistir
final_response = None
iteration = 0

with Live(console=console, refresh_per_second=4) as live:
    while iteration < 50:
        iteration += 1
        live.update(Text("■ pensando...", style="dim"))

        resp = adapter.call(messages)
        messages.append(adapter.assistant_msg(resp))
        final_response = resp

        if resp.done:
            break

        results = []
        for tc in resp.tool_calls:
            preview = tc.inputs.get("command",
                                   tc.inputs.get("path", " ")[:70])
            live.update(Text(f"■ {tc.name}: {preview}", style="cyan"))

            result = execute_tool(tc.name, tc.inputs, project_path)
            results.append(result)

            console.print(f" [dim cyan]{tc.name}[/dim cyan] [dim]{preview}[/dim]")

        messages.extend(adapter.tool_results_msg(resp.tool_calls, results))

# Persiste run e findings no SQLite
# O agente extrai findings do relatorio - aqui usamos placeholder vazio
# Em v0.2: parsear o devguard-report.md para extrair findings estruturados
save_run(
    project_path=project_path,
    run_id=run_id,
    model=config.model,
    stack="detected", # TODO: extrair do relatorio
    findings=findings_extracted
)
mark_fixed_since(project_path, run_id)

# Imprime relatorio final
if final_response and final_response.text:
    console.print(Markdown(final_response.text))

console.print(f"\n[green]Analise concluida.[/green]")
if save_report:
    console.print(f"[dim]Relatorio: {project_path}/devguard-report.md[/dim]")
    console.print(f"[dim]Historico salvo em: {project_path}/.devguard/devguard.db[/dim]")

```



```

import click
from pathlib import Path
from rich.console import Console
from .config import load_config
from .auth import validate_license, AuthError
from .agent import run_agent
from . import __version__

console = Console()

@click.group()
def main():
    """DevGuard – autonomous AI security agent for your codebase."""
    pass

@main.command()
@click.argument("path", default=".", type=click.Path(exists=True))
@click.option("--model", default=None,
              help="Override do modelo. Ex: gpt-4o, claude-sonnet-4-20250514")
@click.option("--no-save", is_flag=True,
              help="Nao salvar relatorio em arquivo")
@click.option("--no-memory", is_flag=True,
              help="Ignorar historico de runs anteriores")
def security(path, model, no_save, no_memory):
    """Executa analise de segurança completa no projeto."""
    project_path = str(Path(path).resolve())
    try:
        config = load_config(model_override=model)
        validate_license(config)
    except AuthError as e:
        console.print(f"[red]Erro de autenticacao:[/red] {e}")
        raise SystemExit(1)
    run_agent(
        agent="security",
        project_path=project_path,
        config=config,
        save_report=not no_save
    )

@main.command()
@click.argument("path", default=".", type=click.Path(exists=True))
def history(path):
    """Exibe historico de analises do projeto."""
    from .memory import get_stats, _connect, init_db
    project_path = str(Path(path).resolve())
    init_db(project_path)
    stats = get_stats(project_path)
    console.print(f"[bold]Historico:[/bold] {project_path}")
    console.print(f"  Analises realizadas: {stats['total_runs']}")
    console.print(f"  Criticos abertos:      {stats['open_critical']}")

```

```
console.print(f"  Altos abertos:      {stats['open_high']}")
console.print(f"  Corrigidos total:    {stats['fixed_total']}")

@main.command(name="version")
def show_version():
    """Exibe a versao instalada."""
    console.print(f"devguard {__version__}")
```

.env.example

bash

```
# Licenca DevGuard – obtenha em https://guardion.ai/devguard
DEVGUARD_API_KEY=dg_...

# API key do LLM (escolha um):
ANTHROPIC_API_KEY=sk-ant-... # Claude Sonnet – recomendado
# OPENAI_API_KEY=sk-...      # GPT-4o
# GROQ_API_KEY=gsk-...       # Llama 3.1 70B (mais barato)

# Para desenvolvimento local:
# DEVGUARD_VALIDATE_URL=http://localhost:8000/api/validate-key
```

README.md

Markdown

```
# DevGuard
Autonomous AI security agent for your codebase.

## Instalacao
pip install devguard

## Configuracao
export DEVGUARD_API_KEY=dg_...      # sua licenca
export ANTHROPIC_API_KEY=sk-ant-... # ou OPENAI_API_KEY / GROQ_API_KEY

## Uso
devguard security ./meu-projeto      # analise completa
devguard security .                  # diretorio atual
devguard security . --no-save        # so imprime, nao salva
devguard history ./meu-projeto       # historico de analises
devguard version

## Memoria entre analises
O DevGuard lembra dos findings entre runs.
Na segunda analise em diante, o agente compara com o historico e diz:
- O que foi corrigido desde a ultima analise
- O que ainda esta aberto (e ha quantos dias)
- O que e novo neste run

O historico fica em .devguard/devguard.db na raiz do projeto.
Adicione .devguard/ ao seu .gitignore.

## Dependencias (opcionais, via Docker)
owasp/zap2docker-stable | aquasec/trivy
projectdiscovery/nuclei | returntocorp/semgrep
```


0 ANTES DE TUDO: ler o backend existente (Passo 0)

Executar o Passo 0 da Seção 2. Ler o código real dos dois projetos. Documentar o contrato exato do endpoint de validação. SÓ ENTÃO começar a implementar.

1 Criar estrutura de pastas e pyproject.toml

Criar `agent_security/` com todos os arquivos vazios. Testar: `pip install -e .` e verificar que o comando 'devguard' aparece no terminal.

2 Implementar config.py

Testar com cada variável de ambiente separadamente. Verificar `SystemExit` quando nenhuma key existe.

3 Implementar auth.py com o contrato real

Usar o que foi encontrado no Passo 0. Testar com `DEVGUARD_VALIDATE_URL=http://localhost:8000/...` Testar key válida, key inválida e sem conexão.

4 Implementar tools.py

Cada função (`_run`, `_read`, `_ls`, `_http`, `_write`) com teste unitário. Testar truncagem de outputs grandes. Verificar `IGNORE_DIRS` no `list_dir`.

5 Implementar memory.py

Testar `init_db()`, `save_run()`, `load_previous_context()` e `mark_fixed_since()`. Rodar dois saves seguidos e verificar que o diff funciona. Verificar que `.devguard/.gitignore` é criado automaticamente.

6 Implementar adapter.py

Testar com Anthropic primeiro. Verificar `tool_results_msg` para cada provider. Testar com uma tool call simples antes do loop completo.

7 Implementar prompts.py

Copiar o `SECURITY_SYSTEM_PROMPT` completo da Seção 8. Não editar sem validar impacto no comportamento.

8 Implementar agent.py

Testar com projeto Flask/FastAPI simples. Verificar que o contexto anterior é injetado no segundo run. Verificar limite de 50 iterações.

9 Implementar cli.py e testar end-to-end

`pip install -e .` e rodar `devguard security` em projeto de teste. Verificar `devguard-report.md` gerado. Verificar `devguard history` funcionando.

Testar localmente antes de publicar

```
bash

# Na pasta agent_security/
pip install -e .                # instala em modo desenvolvimento
devguard version                # deve mostrar 0.1.0

# Configurar variáveis
export DEVGUARD_API_KEY=dg_test_key
export ANTHROPIC_API_KEY=sk-ant-...

# Testar em projeto de exemplo
devguard security ./projeto-de-teste

# Verificar outputs
ls ./projeto-de-teste/devguard-report.md    # relatorio gerado
ls ./projeto-de-teste/.devguard/devguard.db # banco de historico

# Rodar segunda vez para testar memoria
devguard security ./projeto-de-teste        # deve mostrar historico
devguard history ./projeto-de-teste         # estatisticas
```

Publicar no PyPI

```
# 1. Instalar ferramentas de build
pip install hatch twine

# 2. Build
cd documents/guardion-ai/backend/agent_security
hatch build
# Gera dist/devguard-0.1.0-py3-none-any.whl
#     dist/devguard-0.1.0.tar.gz

# 3. Criar conta em pypi.org se nao tiver
# Gerar API token em: pypi.org -> Account settings -> API tokens

# 4. Publicar
twine upload dist/*
# Pedira username e password:
#   username: __token__
#   password: pypi-AgEI... (o token gerado no passo anterior)

# 5. Verificar
pip install devguard # em outra maquina ou venv limpo
devguard version

# Para atualizacoes:
# 1. Incrementar version em pyproject.toml e __init__.py
# 2. hatch build
# 3. twine upload dist/*
```

Dica: testar sem publicar no PyPI real

Use o Test PyPI para validar antes do PyPI real:

```
twine upload --repository testpypi dist/*
pip install --index-url https://test.pypi.org/simple/ devguard
```

Conta separada em test.pypi.org — grátis e sem risco.