Sprint 1 — Project Kickoff & Foundation (4 hrs)

Objective

Stand up the project skeleton, core infra, collaboration workflow and a minimal end-to-end “hello world” scan so the team can iterate safely.

Tasks

Repository & collaboration

- Create monorepo (or two repos) structure: `backend/`, `frontend/`, `infra/`, `docs/`.
- Add `README.md` root with quick start and sprint plan.
- Create issue templates, PR template, branch strategy (main/dev/feature/*).
- Create Trello/Jira board sprint 1 epic + backlog import.

Backend (FastAPI)

- Initialize FastAPI project with Uvicorn, Pydantic models, alembic or migration note (if needed).
- Add `.env` handling and example `.env.example`.
- Implement minimal `/health` and `/version` endpoints.
- Implement a minimal POST `/scan` that accepts `{ "domain": "example.com" }` and returns mock asset JSON (no external collectors yet).
- Add basic logging and structured logs.

Database & infra

- Provision local MongoDB dev instance (docker-compose) and add `docker-compose.yml` with `backend` and `mongo` services.
- Add initial Mongo schema (Pydantic + example Mongo collection setup).
- Add simple config for Render (or note for production deploy).

CI / Tests

- Add GitHub Actions workflow to run backend unit tests and lint on PR.
- Create a basic unit test for `/scan` and `/health`.

Frontend (React + Vite)

- Initialize React + Vite project with Tailwind CSS (Tailwind CLI config).
- Add a minimal page with Login placeholder and a “Trigger Scan” form that calls POST `/scan` to show mock results.
- Add environment configs for API base URL.

Security & Auth (initial)

- Add JWT stub and user model with roles placeholder (no full auth flow yet).
- Create docs entry: security checklist for later sprints.

API Endpoints (initial)

- `GET /health` — returns `{ "status": "ok" }`
- `GET /version` — returns app version
- `POST /scan` — accepts `{ "domain": "example.com" }`, returns mock asset list

- `GET /assets/{doma v in}` — returns mock assets for development

Deliverables

- Working monorepo with `backend/` and `frontend/`.
- Docker-compose dev environment (FastAPI + Mongo).
- CI pipeline that runs tests on PR.
- Minimal working frontend calling backend.
- README with setup & how to run locally.
- Sprint 1 retrospective notes and backlog refinement for sprint 2.

Acceptance Criteria

- `docker-compose up` boots backend and Mongo locally.
- `/health` and `/version` return expected responses.
- `POST /scan` returns a properly shaped JSON (matches your later schema) using mock data.
- Frontend can trigger `POST /scan` and display results.
- GitHub Actions runs unit tests and prevents merge if tests fail.

🎯 Sprint 2 — Core OSINT & Asset Discovery (4 hrs)

Objective
Build and deploy the backend that discovers and catalogs exposed assets.

Tasks
- Develop the FastAPI backend and connect it to MongoDB.
- Implement collectors using open APIs and tools:
  - Subdomain discovery → amass, subfinder, crt.sh API
  - Port/service enumeration → Censys API
  - DNS, WHOIS, and SSL certificate enrichment
- Store collected data with schema:

```json
{
  "domain": "example.com",
  "ip": "192.168.1.1",
  "ports": [80, 443],
  "tech_stack": ["nginx", "react"],
  "ssl_issuer": "Let's Encrypt",
  "last_seen": "2025-10-30T12:00:00Z",
  "source": "censys"
}
```

API Endpoints

POST /scan — trigger OSINT discovery for a domain

GET /assets/{domain} — list discovered assets

GET /stats — summarize total scans, ports, and assets

Deliverables

Async collector modules (async/await)

Unit tests for collectors

Example JSON responses

README.md with environment setup instructions

🧮 Sprint 3 — ML Risk Scoring & Analytics (4 hrs)

Objective

Add a machine-learning risk-scoring engine and analytics dashboard.

Backend Tasks
Integrate scikit-learn risk-scoring module into FastAPI.

Model inputs:

Number of open ports

Outdated software versions

SSL/TLS expiry days

Exposure category (open bucket, code leak, credential)

Breach history (HaveIBeenPwned API)

Train Logistic Regression or XGBoost to produce:

json
Copy code
{
  "domain": "example.com",
  "risk_score": 87,
  "factors": ["open_port_22", "expired_cert"]
}
New API Routes
GET /risk/{domain} — current risk score and factors

GET /risk/trends — risk trend for charts

Frontend (React + Vite)
Build dashboard with TailwindCSS and Recharts

Visualizations:

Risk timeline graph

Top risky assets list

Filter by risk level (low/medium/high)

Add Google sign in
Support multi-tenant model (user ↔ assets)

Deliverables
React + Vite frontend hosted on Vercel

Connected REST API

docs/model.md — model training workflow and dataset schema

⚙ Sprint 4 — Automation + Micro-SaaS Delivery Objective (4 hrs)
Convert the working prototype into a production-ready Micro-SaaS.

Backend Enhancements
Add Redis for scheduled daily rescans.

Integrate Stripe for subscription billing (free + pro tiers).

Implement RBAC (Admin, Analyst, User).

Add Email alerts when risk_score rises > 20 % in 24 h.

Frontend Enhancements
Introduce Workspaces: each = one client/team.

Add usage dashboard (API calls, scan credits, payments).

Create billing page with Stripe Checkout integration.

Add notification center (risk alerts, billing reminders).

Configure GitHub Actions for CI/CD (auto-deploy on push).

Optimize cloud cost: Render (backend) + Vercel (frontend).

Final Deliverables
Production-ready FastAPI + MongoDB backend

Deployed React + Vite frontend with JWT auth

Stripe billing automation

Email alerting system

Comprehensive README.md (install + API docs + screenshots)

System architecture diagram

Sprint 6 — Polish, Hardening, Monitoring & Handover (4 hrs)

Objective

Polish UX, harden production systems, add monitoring/alerts, perform final QA and prepare handoff (docs + demo). This is the final polishing sprint you described.

Tasks

Final QA & Bugfixes

- Run through checklist of open issues; prioritize and fix high/critical bugs.
- Full end-to-end test of scan → risk scoring → alerts → UI workflows.
- Accessibility smoke-check (keyboard navigation, contrast).

Security & Hardening

- Rotate secrets: ensure env secrets are stored in provider vault (Render/Vercel/Secrets).
- Harden JWT (shorter expiry, refresh tokens) and ensure RBAC rules enforced for protected routes.
- Add rate-limiting to public endpoints (IP-based throttling).
- Run dependency vulnerability scan (e.g., `npm audit` / `pip-audit`) and patch critical items.

Observability & Monitoring

- Integrate logs with a hosted log provider or configure centralized logs (stdout structured JSON).
- Add basic metrics endpoints and integrate with monitoring (Prometheus exporter / third-party) — at minimum expose `/metrics` or add request-latency logs.
- Configure alerting rules: email/SMS/Slack when:
    - API error rate > threshold
    - Background scan worker fails
    - Redis or Mongo unreachable
- Add health checks for services and readiness probes.

Performance & Scalability

- Run lightweight load test for core endpoints (e.g., 100 concurrent requests) and document results.
- Add Redis caching for expensive endpoints (e.g., `GET /risk/{domain}`) with TTL.
- Add daily rescans scheduling checks (verify Redis jobs run).

Operational

- Finalize CI/CD: GitHub Actions workflow for main branch deploys to Render (backend) and Vercel (frontend).
- Configure Canary/preview deployments (if available) or set deployment protection rules.

Documentation & Handoff

- Complete `README.md` (install, env, API docs, screenshot gallery).

- Add `docs/model.md` (training steps, dataset schema, feature engineering pipeline).
- Add runbook: how to restart services, how to regenerate API keys, how to revoke Stripe keys, how to onboard a new tenant.
- Prepare a 10–15 minute demo script and short slide deck: architecture diagram, core flows, how to run locally, known limitations, roadmap.
- Create a short post-mortem / lessons learned doc and backlog for future improvements.

Deliverables

- Production-ready deploys (backend + frontend) with CI/CD verified.
- Monitoring + alerting configured and tested.
- Completed documentation: README, API docs, model.md, runbook.
- Demo slide deck and recorded demo (optional — if time allows record a 5–10 min clip).
- Release notes and changelog.

Acceptance Criteria

- All critical/high bugs resolved or logged with mitigation plan.
- Monitoring and at least two alert rules are firing test alerts successfully.
- Redis caching and daily-rescan jobs verified in staging.
- Documentation covers setup, deployment, and operational runbook.
- Demo prepared and walkthrough tested locally.