Architectural Requirements

Quality Requirements Addressed: Link

Architectural Patterns

In developing the BRAD (Bot to Report Abusive Domains) system, several architectural styles have been deliberately chosen to support the project's critical quality requirements, including security, compliance, scalability, maintainability, reliability, and usability. These styles provide the foundation for a modular, secure, and extensible system suitable for automated domain investigations and forensic analysis.

Event-Driven Architecture (EDA)

The BRAD system leverages an event-driven architecture to enable high-throughput, asynchronous workflows. Events such as "domain submitted", "scrape completed", and "AI analysis triggered" are dispatched and consumed by decoupled services.

Quality Requirements Addressed:

- **Scalability**: New event consumers can be added horizontally to meet increased demand during peak investigation periods.
- **Performance**: Asynchronous processing enables faster throughput for multiple domain investigations running in parallel.
- **Reliability**: Persistent queues and event logs allow retries and recovery if services fail mid-process.

Service-Oriented Architecture (SOA)

All major features of the BRAD system are broken down into distinct services such as ScrapeService, AnalyzeService, and ReportService. These services communicate over well-defined REST APIs and follow loose coupling principles.

Quality Requirements Addressed:

- **Scalability**: Each service can be scaled based on load without affecting the others.
- **Maintainability**: Services can be independently updated or replaced, supporting long-term evolution.
- **Interoperability**: Services follow standard formats and protocols (e.g., JSON, HTTP), enabling integration with external threat intel APIs.

Microservices Architecture

BRAD extends SOA into microservices, containerizing each component using Docker. Every major function—scraper bots, ML classifiers, data ingestion pipelines, dashboards—is deployed as an independent container/service.

Quality Requirements Addressed:

- **Scalability**: Each microservice (e.g., a scraping worker) can be replicated independently for load distribution.
- **Maintainability**: Faults or updates are isolated to a single service, minimizing system-wide impact.
- **Portability**: Docker containers ensure that services run reliably across different environments (development, testing, deployment).

Client-Server Model

The BRAD system employs a traditional client-server model. The frontend web dashboard (client) communicates with backend services (server) through RESTful API endpoints.

Quality Requirements Addressed:

- **Usability**: A clear separation between UI and backend enables focused UX design for investigators and general users.
- **Security**: The server centralizes sensitive operations, enforcing access control and API authentication.
- **Compliance**: Client submissions are sanitized and validated on the server to meet data protection regulations.

Layered Architecture

The system is structured in layers, promoting a clear separation of responsibilities:

- **Presentation Layer**: User interfaces and dashboards
- **Application Layer**: Request routing, user sessions
- **Business Logic Layer**: Domain analysis, risk scoring, AI decisions
- **Data Layer**: Storage for logs, reports, submissions

Quality Requirements Addressed:

- **Maintainability**: Developers can make changes to one layer (e.g., UI) without affecting others.
- **Security**: Sensitive operations are encapsulated in deeper layers, reducing attack surface.
- **Reliability**: Layered isolation makes failures easier to contain and debug.

Pipe and Filter Pattern

Data flows through a series of processing components (filters), each performing a specific task in the investigation pipeline:
Scrape → Detect Malware → AI Risk Analysis → Metadata Logging → Report Generation

Quality Requirements Addressed:

- **Maintainability**: Each step can be updated or replaced independently.
- **Reliability**: The pipeline can resume at failed steps without reprocessing the entire chain.
- **Performance**: Processing is streamlined through well-defined input/output interfaces.

Model-View-Controller (MVC) (Frontend – Optional)

The web dashboard for investigators may adopt the MVC pattern, where:

- **Model** manages data (e.g., domain reports, threat levels)
- **View** renders the interface (e.g., graphs, tables)
- **Controller** handles user inputs (e.g., "investigate", "override AI")

Quality Requirements Addressed:

- **Usability**: MVC supports responsive, interactive UIs.
- **Maintainability**: Clearly separated concerns improve code modularity and testability.

Design Patterns

1. Factory Pattern
- **Use Case**: Creating different types of bot agents or report objects depending on domain content (e.g., malware, phishing, scam).
- **Benefit**: Encapsulates object creation, improves scalability when new domain types are introduced.

2. Strategy Pattern
- **Use Case**: Switching between scraping techniques (e.g., simple scraper vs. headless browser) or classification models.
- **Benefit**: Makes it easy to plug in new algorithms or scraping methods without altering the core logic.

3. Observer Pattern
- **Use Case**: Real-time alerting system—notify investigators when a high-risk domain is flagged.
- **Benefit**: Decouples alert logic from the classification engine.

4. Singleton Pattern
- **Use Case**: Global configuration manager (e.g., for API keys, ML model paths, threat intelligence feeds).
- **Benefit**: Ensures a single point of configuration and avoids conflicting settings.

5. Decorator Pattern
- **Use Case**: Enriching domain reports dynamically with new metadata like threat score, WHOIS, SSL info, etc.
- **Benefit**: Adds functionality without modifying existing report structures.

6. Command Pattern

- **Use Case**: Encapsulating user actions like "submit report," "analyze domain," "override AI decision" as objects.
- **Benefit**: Supports undo, logging, and replay features.

7. Builder Pattern

- **Use Case**: Constructing complex domain reports step by step (text, screenshots, metadata, scores).
- **Benefit**: Separates construction logic from representation.

8. Chain of Responsibility Pattern

- **Use Case**: Processing a domain through a pipeline (e.g., scraping → analysis → risk scoring → report generation).
- **Benefit**: Each step handles the task it's responsible for or passes it to the next step.

9. Adapter Pattern

- **Use Case**: Integrating with various external threat intelligence APIs or WHOIS lookup tools.
- **Benefit**: Converts incompatible interfaces into one that fits your system.

10. Proxy Pattern

- **Use Case**: For secure access to the scraper bot or AI module (e.g., rate-limiting, authentication).
- **Benefit**: Adds a layer of control and security around sensitive components.

11. **Mediator Pattern**

- **Use Case**: Manages communication between reporters and investigators through an Admin. Reporters submit reports, and the Admin assigns them to available investigators.
- **Benefit**: Prevents direct communication between parties, improves coordination, and keeps the workflow secure and organized.

Constraints

- **Legal & Compliance Risks**: Must comply with GDPR, POPIA.
- **Domain Blocking & Evasion**: Some sites may block scraping; might require headless browsers or IP rotation. Some websites don't want to be automatically scanned or scraped by bots. So they use techniques to block your bot from accessing their content. To work around this tools like Headless browsers and IP rotation may be used. They prevent the bot from being blocked by making it seem like it is a normal user when it is fact not a normal user.
- **False Positives in AI Classification**: May require manual override or verification, i.e. AI might incorrectly flag a safe domain as malicious. Since AI isn't perfect, there's a chance it could make mistakes. That's why you might need a manual override or human verification, where a security analyst or investigator reviews the case and decides if the AI's decision was actually correct.
- **Data Privacy & Ethics**: Need secure storage, anonymization, and ethical data handling practices.

- **Budgetary Limits**: Although a server and some funds are provided, the project must stay within the allocated budget.