Architectural Requirements

Architectural Design Strategy: Design based on Quality Requirements

Making It Easy to Use

Usability is just as important as functionality. We design dashboards and submission pages that are simple and clear so that users don't get confused. This helps reporters submit suspicious domains quickly and investigators work efficiently with the tools we provide.

Building at a Steady Pace

Agile encourages teams to work at a pace they can keep up. By focusing on performance and clean structure, we avoid messy code and make sure the system is efficient and doesn't slow down as it grows. This helps the team avoid burnout and reduces problems later.

Keeping the System Safe and Reliable

Since BRAD works with sensitive information, we always think about security and reliability in our design. We implement strong access control, monitor system activity, and handle failures in a way that doesn't put data or users at risk. This ensures that the system keeps working even under stress.

Collaboration

When we design based on clear quality requirements, everyone on the team knows what matters most. Whether someone is working on the frontend or backend, we all follow the same goals like making the system easy to use, secure, and quick. This makes teamwork smoother and more focused.

Adapting to Change

Cyber-security threats change quickly. Designing around scalability and maintainability helps us make changes without breaking the system. This means we can update the scraper, switch AI models, or add new features easily. It also lets us respond to new ideas or feedback from users as the project goes on.

Step-by-Step Improvements

We build BRAD in small steps. Each sprint adds something new like scraping, AI analysis, or reporting. By keeping each step small and tied to a quality goal (like performance or accuracy), we make sure we're always improving the system in a smart and manageable way.

Customer Trust and Frequent Progress

We want users and stakeholders to feel confident in what we're building. By focusing on quality requirements like security, reliability, and usability, we make sure that every version of the BRAD system is stable, safe, and useful. This helps us deliver real value early and often, which is a key part of Agile.

Architectural Styles

In developing the Bot to Report Abusive Domains (BRAD) system, we have selected several architectural styles to meet key quality requirements, including security, compliance, scalability, reliability, usability, and maintainability. These styles work together to support BRAD's mission: allowing users to report suspicious domains, processing them through an automated analysis pipeline, and presenting accurate and accessible results to investigators.

The Event-Driven Architecture (EDA) enables BRAD to handle multiple domain reports asynchronously. When a suspicious domain is submitted, it triggers a chain of downstream processes such as scraping, malware checking, AI scoring, and report generation. This pattern boosts scalability and performance by decoupling components and allowing them to process in parallel. It also improves reliability by using persistent queues, which ensure that failed tasks can be retried without data loss.

The Gatekeeper Pattern enforces system-wide security, reliability, and compliance. Acting as a single entry point, it performs user authentication, role-based access control (RBAC), and multi-factor authentication (MFA) where needed. The Gatekeeper also filters and validates all incoming requests, applying rate limiting and logging for auditability. This protects backend services from malicious or malformed input and ensures alignment with data protection laws like POPIA and GDPR.

The Client-Server Model separates the frontend user interface from backend processing. Users interact with the system via a web portal by submitting URLs and reviewing reports while the backend handles authentication, analysis, and data storage. This model enhances usability by allowing a focused and responsive interface. It also supports security and compliance by ensuring that sensitive operations are handled server-side under controlled conditions.

The Layered Architecture organizes BRAD into logical layers: the presentation layer (user interface), application layer (API and Gatekeeper), business logic layer (analysis and classification), and data layer (databases and logs). This improves maintainability, as each layer can be updated independently and reliability as issues in one layer don't directly affect others. It also enhances security by restricting access to sensitive data handling within backend layers only.

The Pipe and Filter Pattern structures the backend analysis workflow as a clear processing pipeline: Scrape → Malware Detection → AI Risk Analysis → Report Generation. Each stage is independent, transforming the data and passing it forward. This improves maintainability by allowing individual steps to be replaced or updated. It also enhances reliability by

supporting error handling and recovery at each stage, and improves performance by enabling potential parallel execution of steps.

Event-Driven Architecture (EDA)

Quality Requirements Addressed:

- Scalability: Events trigger decoupled processes that scale independently.
- Performance: Supports real-time, asynchronous execution.
- Reliability: Persistent queues allow task recovery after failure.

Gatekeeper Pattern

Quality Requirements Addressed:

- Security: Validates and authenticates all incoming traffic.
- Reliability: Shields services from overload and malformed input.
- Compliance: Enforces logging and data protection rules (POPIA/GDPR).

Client-Server Model

Quality Requirements Addressed:

- Usability: Frontend provides clear UI for reporters and investigators.
- Security: Backend controls sensitive operations and data.
- Compliance: Ensures data validation and consent handling on the server.

Layered Architecture

Quality Requirements Addressed:

- Maintainability: Each layer can be changed independently.
- Security: Sensitive logic is isolated in protected backend layers.
- Reliability: Faults are contained within individual layers.

Pipe and Filter Pattern

Quality Requirements Addressed:

- Maintainability: Filters can be updated without breaking the flow.
- Reliability: Pipeline resumes from failed stages where possible.
- Performance: Processing steps can be parallelized and optimized.

Architectural Quality Requirements

1. Security (Most Important)

Security is the foundation of the B.R.A.D system, given its handling of sensitive data like user-submitted URLs, forensic metadata, and potentially malicious content. Unauthorized access or breaches could lead to severe consequences such as data leaks, false reports, or misuse of the system for cyber-attacks. Therefore, security controls, encrypted storage, secure APIs, role-based access control, and container isolation must be thoroughly enforced to protect both user and system integrity.

Stimulus Source	**Stimulus**	**Response**	**Response Measure**	**Environment**	**Artifact**
Malicious actors/ Attackers.	Attempt to compromise data or infrastructure.	System should block unauthorized access, validate input and encrypt sensitive information.	100% of sensitive data encrypted at rest and in transit. All **RBAC (Role Based Access Control)** and **MFA (Multi-Factor Authentication)** enforced.	Production environment.	BRAD Backend/ API System

2. Compliance

Compliance ensures that the system operates within the legal and ethical boundaries defined by regulations like GDPR and POPIA. This is especially important for a tool that collects and processes potentially identifiable or legally sensitive data. Compliance includes implementing consent mechanisms, depersonalizing data when possible, logging access to personal data, and providing the right to be forgotten.

Stimulus Source	**Stimulus**	**Response**	**Response Measure**	**Environment**	**Artifact**
Legal/ Regulatory Bodies.	Data privacy and regulatory audits.	System should ensure legal compliance in data handling and provide user data control mechanisms.	**GDPR** and **POPIA** checklists passed; audit logs maintained; user data deletion supported.	Production environment.	Data Processing Components.

3. Reliability

The reliability of B.R.A.D ensures that forensic investigations can be conducted consistently and accurately. The system should gracefully handle failed URL submissions, avoid crashes during analysis, and recover from bot failures without corrupting data. High reliability builds trust in the system's outputs and enables analysts to depend on its results for critical decision-making.

Stimulus Source	**Stimulus**	**Response**	**Response Measure**	**Environment**	**Artifact**
System Users.	Submission of various domains, including malformed or malicious ones.	System should maintain stable operation and report errors clearly.	99.9% uptime, bot recovers from crashes within 60 seconds.	Production environment.	Bot Engine and Report System.

4. Scalability

Scalability is essential to support the analysis of many domain reports simultaneously. B.R.A.D must be able to grow with demand, especially during cyber incident spikes. It should process multiple domain submissions concurrently without bottlenecking the system or slowing down analysis pipelines. By ensuring scalability, the system can maintain optimal performance under high loads, enabling faster processing and quicker turnaround times for forensic results.

Stimulus Source	**Stimulus**	**Response**	**Response Measure**	**Environment**	**Artifact**
Multiple Users.	Submission of multiple links at the same time.	System should scale horizontally to handle multiple concurrent analyses.	Supports 500+ concurrent domain submissions with average analysis < 10s/domain.	Production environment.	Domain Analysis Pipeline.

5. Maintainability

B.R.A.D's architecture must allow for frequent updates such as patching vulnerabilities, integrating new threat intelligence feeds or adapting AI models. The system must be designed with modularity and clear interfaces between components (e.g., scrapers, AI, storage) so developers can make targeted changes without affecting the whole system.

Stimulus Source	Stimulus	Response	Response Measure	Environment	Artifact
Development Team.	Requirement to update scraping logic or AI model.	System should allow modular, low-risk updates with minimal downtime.	Docker-based components, automated deployment pipeline, <5 min rollout	Development environment.	Bot Container & AI Modules.

Architectural Design and Pattern

In developing the BRAD (Bot to Report Abusive Domains) system, several architectural patterns have been chosen to support the project's critical quality requirements.

Gatekeeper Pattern

The Gatekeeper Pattern is implemented in BRAD as a dedicated security layer that mediates all incoming traffic to the system. This component acts as a centralized entry point that handles user authentication, enforces role-based access control (RBAC), and verifies that each request meets the system's security and compliance policies. By introducing this pattern, BRAD directly addresses security, reliability, and compliance requirements. Unauthorized or malformed requests are blocked before reaching sensitive backend services such as the scraper or AI classifier. The Gatekeeper also integrates multi-factor authentication (MFA) for investigator accounts, further securing access to forensic data. In high-traffic or adversarial scenarios, it supports rate limiting, input sanitization, and logging to mitigate threats such as denial-of-service (DoS) attacks or injection attempts.

Quality Requirements Addressed:

1. **Security**: All requests are authenticated, authorized, and validated before reaching internal services, preventing unauthorized access and injection attacks.
2. **Reliability**: The Gatekeeper handles rate limiting, failover routing, and input filtering to protect internal services from overload or failure.
3. **Compliance**: Every access attempt is logged and checked against regulatory rules, ensuring adherence to GDPR and POPIA obligations.

Event-Driven Architecture (EDA)

The Event-Driven Architecture (EDA) enables BRAD to process large volumes of domain investigation requests by allowing system components to operate asynchronously in response to discrete events. When a user submits a suspicious URL, this event triggers a pipeline of subsequent actions such as scraping, malware scanning, AI-based risk scoring, and report generation each executed by specialized services. This pattern enhances scalability by enabling horizontal scaling of event consumers, allowing the system to handle multiple investigations concurrently. It also improves performance by decoupling producers (e.g., the submission module) from consumers (e.g., the scraper bot), enabling real-time, non-blocking execution. Reliability is also addressed through persistent event logs, which ensure that failed or interrupted processes can be recovered and replayed without data loss.

Quality Requirements Addressed:

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

Service-Oriented Architecture (SOA)

The Service-Oriented Architecture (SOA) is used to decompose BRAD into modular services such as Scrape-Service, Analyse-Service, and Report-Service, each responsible for a well-defined function. These services interact via standardized RESTful APIs, allowing them to operate independently and be developed, deployed, or scaled without disrupting the system as a whole. This directly improves maintainability, as updates or bug fixes in one service do not cascade into others. It also improves interoperability, enabling future integration with external systems such as threat intelligence databases or registrar reporting interfaces. Moreover, scalability is enhanced by the ability to scale only the services under load (e.g., multiple scraper instances during high-volume submissions) rather than the entire system.

Quality Requirements Addressed:

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

Client-Server Model

The Client-Server Model is employed in BRAD to separate the frontend interfaces (client) from backend processing (server). The frontend includes the public user submission portal and the investigator dashboard, both of which communicate with backend services over secured APIs. This pattern strengthens security, as all critical logic and sensitive data processing are centralized on the server, where access can be tightly controlled. It also supports compliance by allowing the server to enforce data validation, consent mechanisms, and logging in accordance with regulations like POPIA and GDPR. Additionally, the model enhances usability, as the client can be optimized for user experience without compromising backend integrity.

Quality Requirements Addressed:

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

Layered Architecture

The system adopts a Layered Architecture to structure its internal logic into four distinct layers: the presentation layer (UI), application layer (API gateway and authentication), business logic layer (scraping and risk analysis), and data layer (databases and logs). This separation improves maintainability, because changes in one layer (e.g., updating the UI) do not ripple across unrelated layers. It also supports security by isolating sensitive logic in backend layers that are not exposed to users. Furthermore, reliability is strengthened, as each layer is testable in isolation, reducing the likelihood of cascading failures.

Quality Requirements Addressed:

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

Pipe and Filter Pattern

The Pipe and Filter Pattern underpins BRAD's core investigation pipeline, where 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

Each component (filter) transforms the input and passes it along the pipeline. This modular design improves maintainability, as filters can be added, replaced, or removed without redesigning the entire flow. It also improves reliability by allowing error handling and fall-back mechanisms at each stage. Additionally, performance benefits from clearly defined processing stages, which can be parallelized or scaled independently when needed.

Quality Requirements Addressed:

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

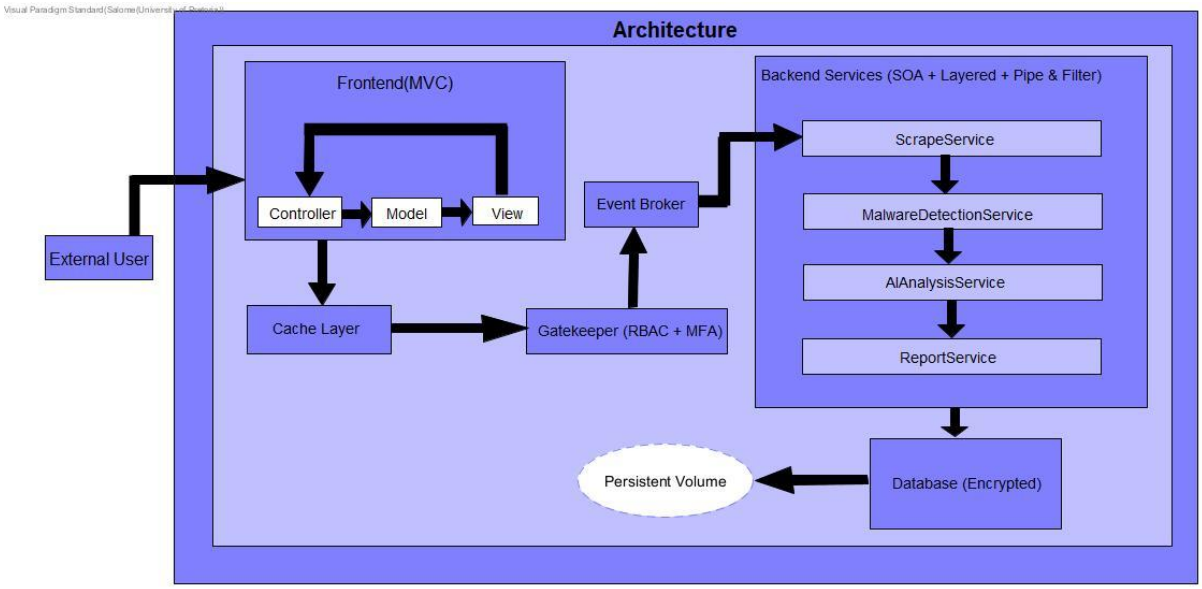
Model-View-Controller (MVC)

On the frontend, the Model-View-Controller (MVC) pattern is applied to the investigator dashboard to cleanly separate concerns. The model holds domain data and system state, the view renders the UI (e.g., graphs, logs, alerts), and the controller handles user input and orchestrates responses. This structure enhances usability by ensuring that the interface is responsive and intuitive. It also improves maintainability, as frontend developers can update visual components, logic, or data handling independently reducing the likelihood of bugs and simplifying testing.

Quality Requirements Addressed:

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

Together, these architectural patterns form a unified blueprint for BRAD's development. Each pattern was chosen not only for technical elegance, but for its direct and measurable impact on the system's critical quality requirements. This ensures that BRAD is not only functional but secure, adaptable, and resilient in the face of ever-evolving cyber-security threats.



Design Patterns

Chain of Responsibility

As a group, we have chosen the Chain of Responsibility design pattern as the fundamental approach to how the BRAD (Bot to Report Abusive Domains) system is architected. This pattern aligns naturally with the system's core investigation pipeline, which follows a sequential flow: **Submission → Scraping → Malware Detection → Metadata Extraction → AI Risk Scoring → Report Generation**. Each stage in this chain represents a specialized processing component responsible for a specific task in the analysis workflow. By structuring BRAD's domain processing in this way, we ensure that data moves through clearly defined steps, allowing each module to operate independently while contributing to the overall forensic report generation.

We selected the Chain of Responsibility pattern because it provides clear separation of concerns, modularity, and flexibility, which are essential qualities for a cyber-security system that must evolve alongside new threats and technologies. This pattern allows us to insert, remove, or replace components in the processing chain (e.g., switching to a new AI model or adding a threat enrichment step) without disrupting the entire system. It also enhances maintainability, since each component is self-contained and testable in isolation, and improves reliability by enabling fine-grained error handling and fall-back mechanisms at each stage. Most importantly, it mirrors the linear and asynchronous nature of BRAD's domain analysis pipeline, making it a natural and effective fit for the architecture.

Architectural Constraints

Adherence to Legal Standards and Regulations:

GDPR and POPIA Compliance

The BRAD system must enforce strong data protection measures to comply with the General Data Protection Regulation (GDPR) and the Protection of Personal Information Act (POPIA). This includes encrypting sensitive data during submission, storage, and transmission, and ensuring that user consent is obtained before data is processed.

Audit Logging and Data Traceability

To support compliance and forensic analysis, the system must maintain immutable logs of all critical events, including domain submissions, analysis results, and investigator actions. These logs must be securely stored and accessible for auditing without exposing sensitive information.

Access Control for Sensitive Operations

Role-based access control (RBAC) must be implemented to restrict access to investigation tools and domain intelligence data. Investigators and admins must have different permissions than regular users, with multi-factor authentication required for elevated access.

Balancing Automation and Human Oversight:

Human-in-the-Loop Decision Making

BRAD must allow investigators to review and override automated AI decisions, especially in cases where a domain is incorrectly flagged. This means the architecture should support both automated and manual review processes in a seamless workflow.

Technical Limitations and Deployment Constraints:

Budget and Infrastructure Limitations

The system must run within the limits of the provided server resources. This constraint affects how services are deployed and scaled, requiring the use of lightweight containers and efficient background processing to avoid memory and CPU overuse.

Network Restrictions for Scraping

Some domains may block automated scraping using bot detection methods. To handle this, the architecture must support scraping strategies such as headless browsers, IP rotation, and custom user-agents to avoid detection and ensure data can still be collected.

False Positives in AI Classification

AI models are not perfect and may misclassify legitimate domains. The architecture must therefore include a manual verification layer and support model retraining or adjustment without disrupting the rest of the system.

Ethical and Data Handling Considerations:

Data Anonymization and Minimization

User-submitted URLs and metadata may include sensitive content. The system must limit the amount of personal data stored and ensure anonymization wherever full identification is not required for investigation purposes.

Transparency and Explainability

The risk classification system should produce results that investigators can understand and explain. This requires AI outputs to include confidence levels, contributing factors, or traceable indicators rather than just final labels.

Technology Choices