Hello, World! Hello, World Inc

B.R.A.D. Demo 1

by Hello World! Inc

Overview

1. Introduction
2. What is B.R.A.D.
3. Requirements
4. Use case diagram
5. Domain model
6. User stories
7. Unit testing
8. Live Demo

Introduction

Our Team:

- E (Ethan) Vletter u22497082 - Project Manager, DevOps
- TS (Tebatso) Mahlathini u22611704 - API, DevOps
- C (Carinda) Smith u22652974 - API
- M (Megan) Pretorius u23708833 - UI
- S (Salome) Kalaka u19364742 - API

Documents link

What is B.R.A.D.

B.R.A.D. (**B**ot to **R**eport **A**busive **D**omains) project, a cybersecurity web application that automates the analysis of potentially malicious URLs.

Users can submit suspicious links through the website, where an AI-powered bot safely visits and analyzes the domain. The system extracts metadata, detects threats like malware, and compiles forensic reports.

Core Requirements

1.1. User Submission Portal
1.2. Scraping & Malware Detection
1.3. Forensic Data Collection
1.4. AI Risk Analysis
1.5. Evidence Submission
1.6. Investigator Dashboard
1.7. Secure Storage

Optional features

2.1. Threat Intelligence Lookup
2.2. Automated WHOIS & DNS
2.3. Domain Similarity Detection
2.4. Real-Time Alerts
2.5. Historical Tracking
2.6. Multi-Language Support

Wow factors

3.1. Live Sandbox Testing
3.2. Machine Learning Risk Scores
3.3. Automated Threat Hunting
3.4. Blockchain Evidence
3.5. Auto Takedown Requests
3.6. Dark Web Checks

Architectural requirements - Architectural Patterns

Gatekeeper Pattern : a dedicated security layer that mediates all incoming traffic to the system.

Quality Requirements Addressed

- Security
- Reliability
- Compliance

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.

Quality Requirements Addressed:

- Scalability
- Performance
- Reliability

Architectural requirements - Architectural Patterns

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.

Quality Requirements Addressed:

- Scalability
- Maintainability
- Interoperability

Architectural requirements - Architectural Patterns

Micro-services Architecture : builds on SOA by containerizing each service using Docker.

Quality Requirements Addressed:

- Scalability
- Maintainability
- Portability

Client-Server Model : is employed in BRAD to separate the frontend interfaces (client) from backend processing (server).

Quality Requirements Addressed:

- Usability
- Security
- Compliance

Architectural requirements - Architectural Patterns

Layered Architecture : The system adopts a Layered Architecture to structure its internal logic into four distinct layers:

1. The presentation layer (UI)
2. Application layer (API gateway and authentication)
3. Business logic layer (scraping and risk analysis)
4. Data layer (databases and logs).

Quality Requirements Addressed:

- Maintainability
- Security
- Reliability

Architectural requirements - Architectural Patterns

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

Quality Requirements Addressed:

- Maintainability
- Reliability
- Performance

Architectural requirements - Architectural Patterns

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.

Quality Requirements Addressed:

- Usability
- Maintainability

Architectural Constraints

- **Legal & Compliance Risks** : Must comply with GDPR, POPIA
- **Domain Blocking & Evasion :** Some sites may block scraping; might require headless browsers or IP rotation
- **False Positives in AI Classification :** May require manual override or verification, i.e. AI might incorrectly flag a safe domain as malicious
- **Data Privacy & Ethics :** Need secure storage, depersonalization, and ethical data handling practices.
- **Budgetary Limits :** Although a server and some funds are provided, the project must stay within the allocated budget.

Service Contracts

Analyze Domain (Internal)

Service Contract Name: `BOT /internal/analyse-domain`

Parameters: { "submissionID": "UUID" }

View Domain Report

Service Contract Name: `GET /api/reports/{domainId}`

Parameters: { "domainId": "UUID" }

Register

Service Contract Name: `POST /api/auth/register`

Parameters: { "name": "string", "email": "string", "password": "string" }

Login

Service Contract Name: `POST /api/auth/login`

Parameters: { "email": "string", "password": "string" }

Submit Suspicious Domain

Service Contract Name: `POST /api/domains/report`

Parameters: { "domain": "string", "evidenceFile": "File (optional)" }

Upload Evidence

Service Contract Name: `POST /api/evidence/upload`

Parameters: { "submissionId": "UUID", "file": "File" }

3 Use Cases

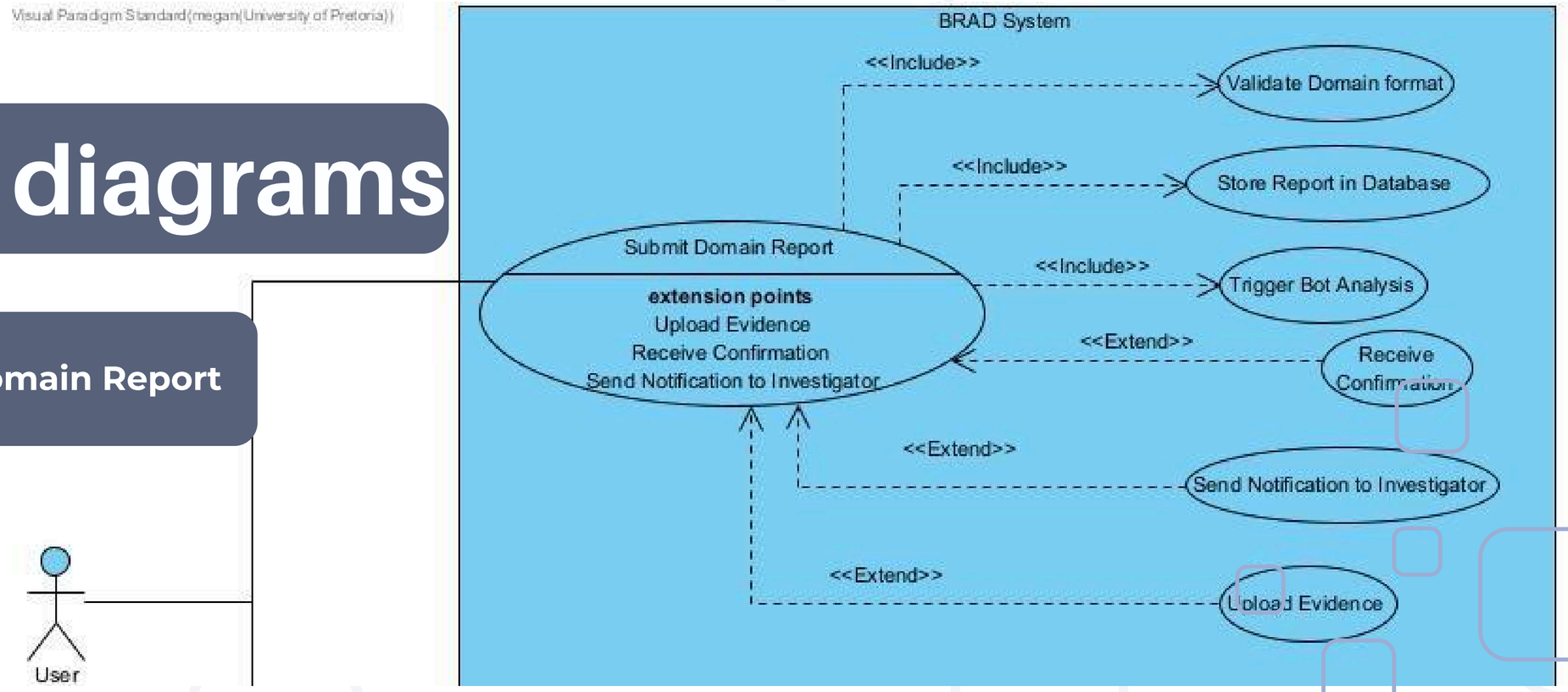
Use Case 1: Submit Domain Report

Use Case2: View Submitted Reports

Use Case 3: Analyse Forensic

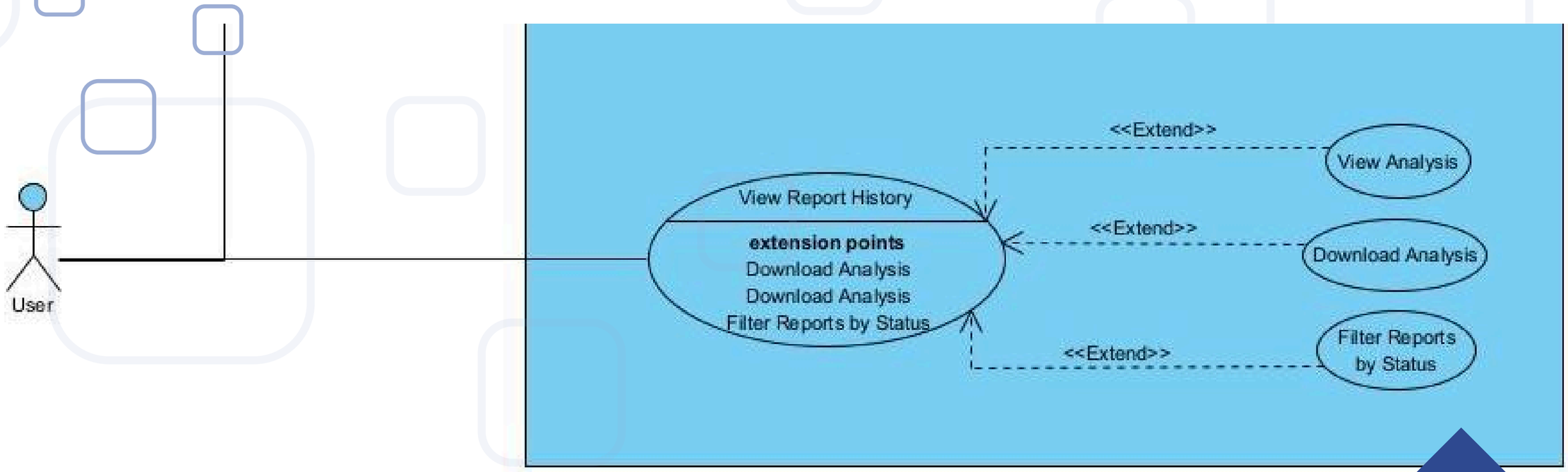
Use case diagrams

Use Case 1: Submit Domain Report



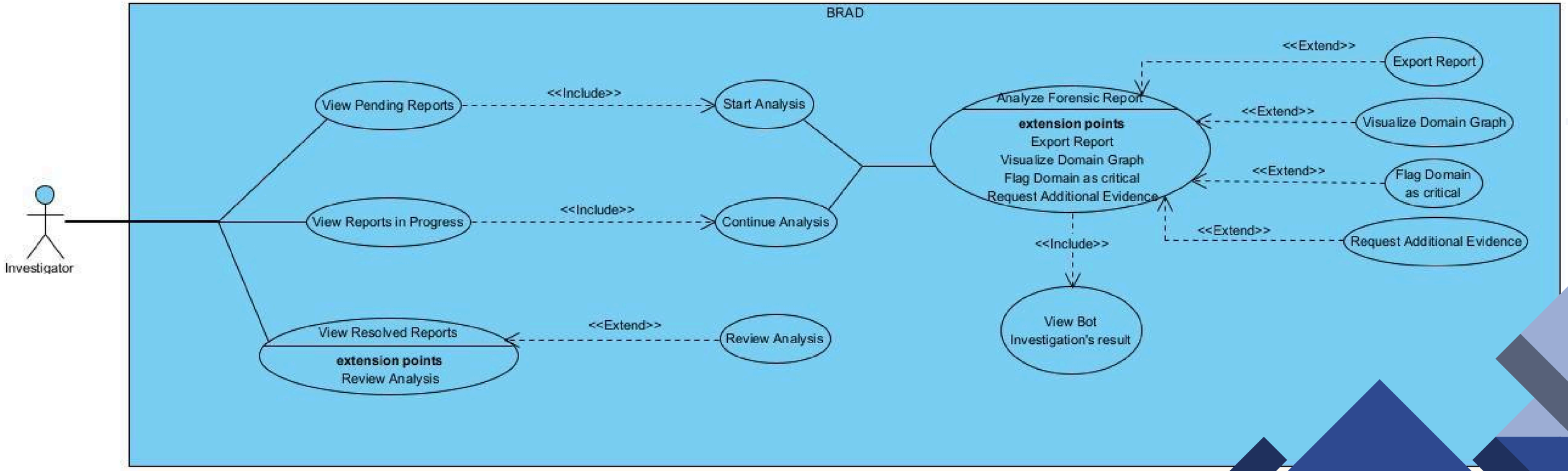
Use case diagrams

Use Case2: View Submitted Reports

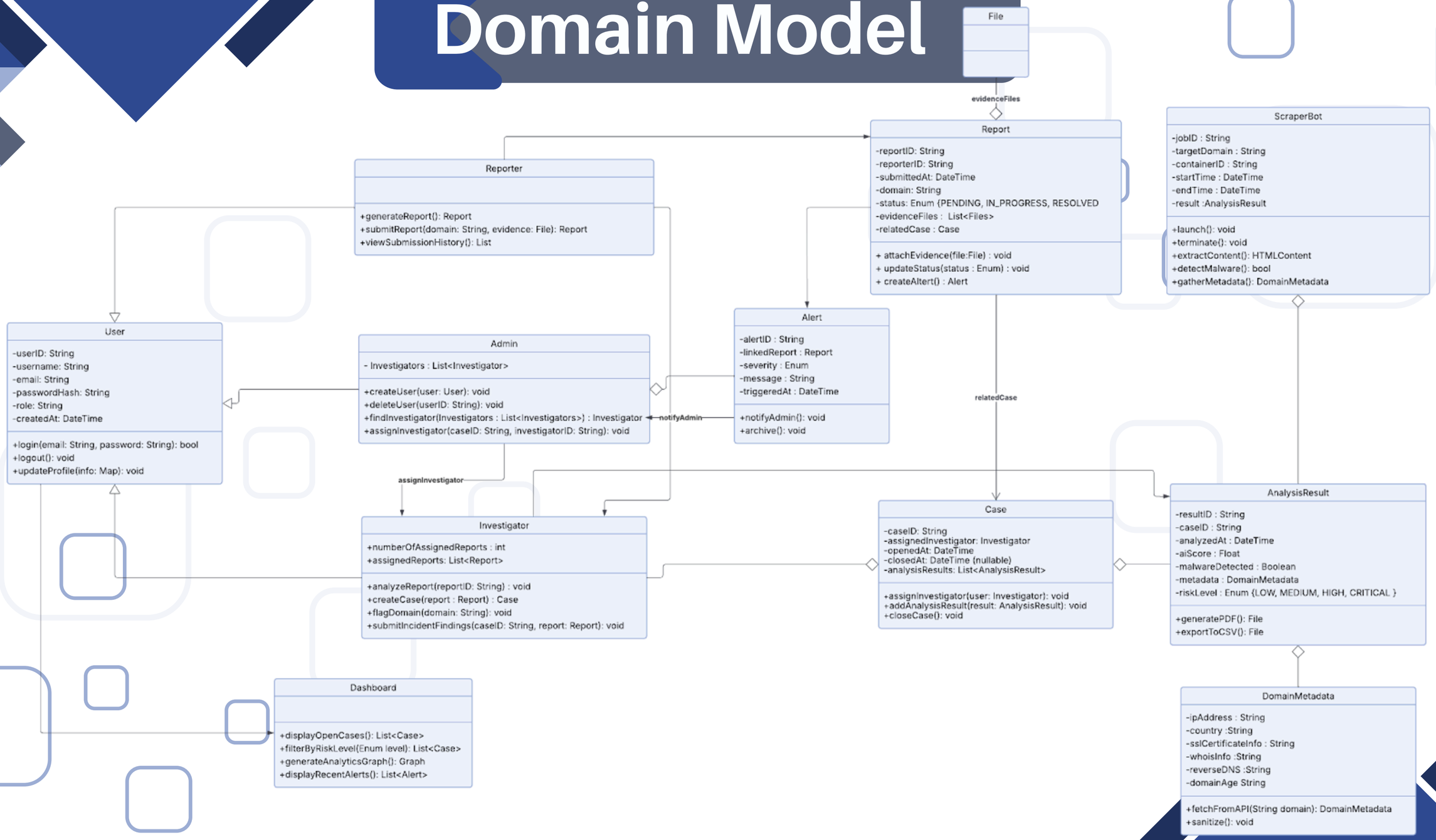


Use case diagrams

Use Case 3: Analyse Forensic



Domain Model



User Stories

1.Role: General user(Reporter)

- - I can submit a suspicious domain via a simple form
- - I can optionally add notes or upload evidence
- - I receive confirmation that my report is submitted
- - I can track my report status and receive feedback

User Stories

2.Role:Investigator

- - I can view all submitted reports
- - I can see risk scores and AI verdicts
- - I can open a detailed report with metadata and evidence
- - I can update the report status
- - I can send feedback to the original reporter

User Stories

3.Role:Admin

- - I can view all registered users
- - I can promote a user to the role of investigator
- - I can demote an investigator to a general user

Unit testing

```
$ npm test

> brad-api@1.0.0 test
> jest --coverage

PASS  tests/report.test.js
PASS  tests/auth.test.js
-----------|---------|----------|---------|---------|-------------------
File       | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------|---------|----------|---------|---------|-------------------
All files  |     100 |      100 |     100 |     100 |
 auth.js   |     100 |      100 |     100 |     100 |
 report.js |     100 |      100 |     100 |     100 |
-----------|---------|----------|---------|---------|-------------------

Test Suites: 2 passed, 2 total
Tests:       11 passed, 11 total
Snapshots:   0 total
Time:        2.722 s
Ran all test suites.
```

```
PS C:\Users\megan\OneDrive\Desktop\Cos 301\Captone\BRAD\BRAD\frontend> npm test

> frontend@0.0.0 test
> jest

PASS  src/__tests__/Landing.test.jsx (6.062 s)
PASS  src/__tests__/InvestigatorDashboard.test.jsx (6.054 s)
PASS  src/__tests__/Register.test.jsx (6.174 s)
PASS  src/__tests__/UserSettings.test.jsx (6.275 s)
  ● Console

    console.log
      Fields to update: { firstName: 'Alice' }

      at log (src/pages/UserSettings.jsx:42:13)

PASS  src/__tests__/Login.test.jsx (6.313 s)
PASS  src/__tests__/ReporterDashboard.test.jsx (6.476 s)

Test Suites: 6 passed, 6 total
Tests:       20 passed, 20 total
Snapshots:   0 total
Time:        20.715 s
Ran all test suites.
```

Swagger

User & Report API 1.0.0 OAS 3.0

Auth

- POST /register Register a new user
- POST /login Login
- POST /logout Logout

Reports

- POST /report Submit a suspicious domain
- GET /reports Get all submitted reports
- GET /forensics/{id} Perform forensic analysis

Bot

- GET /pending-reports Get the next pending report
- POST /analyzed-report Submit analysis result

Investigator

- PATCH /report/{id}/decision Submit investigator verdict
- PATCH /report/{id}/assign Assign report to investigator

Schemas

Bot

GET /pending-reports Get the next pending report

POST /analyzed-report Submit analysis result

Parameters

No parameters

Request body required

Example Value | Schema

```
{
  "id": "string",
  "analysis": {
    "domain": "string",
    "verdict": "string",
    "riskScore": 0,
    "scannedAt": "string",
    "summary": "string",
    "title": "string",
    "ip": "string",
    "registrar": "string",
    "sslValid": true,
    "whoisOwner": "string"
  }
}
```

Responses

Code	Description

Github Repository

```
-> main
    -> dev
        -> ui
        -> docs
        -> api
        -> docker-container
            -> derived branch A
        -> ai-bot
            -> derived branch B
```

- GitHub branching Strategy
- main and dev rules

CI/CD pipeline

Future plans to automatically run tests on contributions using GitHub actions.

These tests include:

- Unit testing
- Integration testing
- End-to-End testing
- Building

Live Demo

Thank You