Software Design Document (SDD)

MEMORY FORENSICS AUTOMATION TOOL FOR MALWARE DETECTION IN RAM

1. Introduction

Purpose-

This Software Design Document (SDD) provides a comprehensive description of the architectural, structural, and technical design of the Memory Forensics Automation System. It translates the functional and non-functional requirements specified in the Software Requirements Specification (SRS) into a detailed implementation-level design.

The document defines:

- System architecture
- Component interaction
- Data design
- Interface specifications
- Security mechanisms
- Deployment architecture
- CI/CD pipeline integration

This document serves as a technical blueprint for development, testing, evaluation, and future system enhancement.

Definitions and acronyms

Full Forms:

SDD – Software Design Document

SRS – Software Requirements Specification

API – Application Programming Interface

UI – User Interface

DB – Database

CI/CD – Continuous Integration / Continuous Deployment

RBAC – Role-Based Access Control

REST – Representational State Transfer

2. References

- Software Requirements Specification (SRS)
- Jenkins Pipeline Documentation
- Node.js Documentation
- React Documentation
- Docker Documentation

3. System Overview

3.1 System Summary

The Memory Forensics Automation System is a web-based interface used for launching automated forensic processes. The system combines a MERN stack application (MongoDB, Express, React, Node.js) with Jenkins CI for managing forensic analysis tasks.

The system supports the following functionalities:

Launching forensic analysis

Execution of automated pipelines

Displaying execution stages

Modular design for easy extension

3.2 High-level goals:

- Automate forensic workflow execution
- Provide CI/CD orchestration using Jenkins
- Ensure modular separation of frontend, backend, and orchestration layers
- Enable scalable and extensible forensic processing design
- Demonstrate DevOps integration within full-stack architecture

4. Architectural Design

4.1 Architectural Overview

The Memory Forensics Automation System follows a layered client–server architecture integrated with a CI/CD orchestration layer. The system is composed of four primary layers:

1. Presentation Layer (React Frontend)
2. Application Layer (Node.js Backend)
3. Orchestration Layer (Jenkins CI Pipeline)
4. Processing Layer (Forensic Analysis Engine – Simulated)

Each layer is designed with clear separation of concerns to ensure modularity, maintainability, and scalability.

- The frontend is responsible only for user interaction.
- The backend handles business logic and API communication.
  Jenkins acts as an orchestration engine.
- The analysis engine performs the forensic workflow (simulated in current implementation).

4.2 Architectural style

The system follows a layered architectural pattern because:

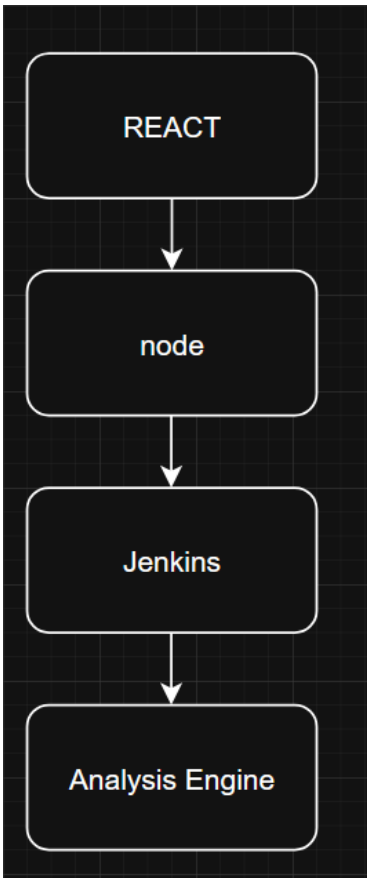
- It separates UI from business logic.
- It allows backend replacement without affecting frontend.
- CI orchestration is isolated from application logic.
- It supports future distributed deployment.

Jenkins was selected as the orchestration tool because:

- It is an industry-standard CI/CD platform.
- It provides stage visualization.
- It supports REST-based triggering.
- It enables scalable workflow automation.

REST-based communication is used because:

- It is stateless.
- It supports language-independent interaction.
- It simplifies integration between Node.js and Jenkins**.**



4.3 Component Architecture

The major system components are:

- React Frontend
- Node.js REST API
- Jenkins Pipeline
- Docker Engine
- Simulated Analysis Engine

Component Interaction Flow

User → Frontend → Backend API → Jenkins → Analysis Engine → Console Output

Each component communicates using HTTP-based REST calls.

4.4 Deployment Architecture

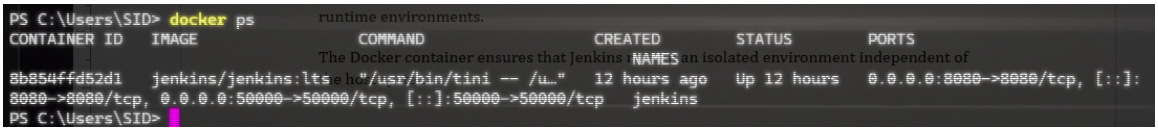
The system is deployed locally for development and evaluation purposes.

Deployment Environment:

- React Frontend → localhost:3000
- Node Backend → localhost:5000
- Jenkins (Docker Container) → localhost:8080
- Docker Engine → Manages Jenkins container

All components operate within the same machine but are logically separated by ports and runtime environments.

The Docker container ensures that Jenkins runs in an isolated environment independent of the host system.



4.5 Data Flow Description

- User clicks “Start Analysis” in frontend.
- Frontend sends HTTP POST request to backend.
- Backend authenticates and triggers Jenkins pipeline.
- Jenkins executes pipeline stages sequentially.
- Execution logs are generated.
- Backend returns success response.
- Frontend displays confirmation message

5. Module / Component Design

5.1 Frontend Module

Purpose

Provides the user interface to initiate forensic workflow and display system responses.

Responsibilities

- Render dashboard.
- Provide “Start Analysis” button.
- Send HTTP POST request to backend.
- Display confirmation message.

Internal Design

The frontend uses functional components with event-driven interaction.
The Fetch API is used to communicate with backend services.

Input

User button click event.

Output

HTTP POST request to backend endpoint `/start-analysis`.

```
app.post("/start-analysis", async (req, res) => {
  try {
    await axios.post(
      "http://localhost:8080/job/forensics-demo/build",
      {},
      {
        auth: {
          username: "sidspipelines",
          password: "1185e65f6e5df638e23a0a40732085d09d"
        }
      }
    );
```

5.2 Backend Module

Purpose

Acts as middleware between frontend and Jenkins orchestration layer.

Responsibilities

- Expose REST API.
- Handle incoming requests.
- Authenticate with Jenkins using API token.
- Trigger Jenkins pipeline.
- Return JSON response.

API Endpoint

- POST /start-analysis

Request Format

No request body required.

Response Format

Success:
{
"message": "Pipeline triggered successfully"
}

Failure:
{
"error": "Failed to trigger pipeline"
}

Internal Processing Logic

1. Receive request.
2. Create HTTP POST call to Jenkins.
3. Attach Basic Authentication header.
4. Await response.
5. Return status to client.

5.3 Jenkins CI Module

Purpose

Coordinates and orchestrates forensic workflow execution.

Pipeline Stages

Stage 1 – Trigger Received
Stage 2 – Simulated Analysis
Stage 3 – Report Generation

Stage Descriptions

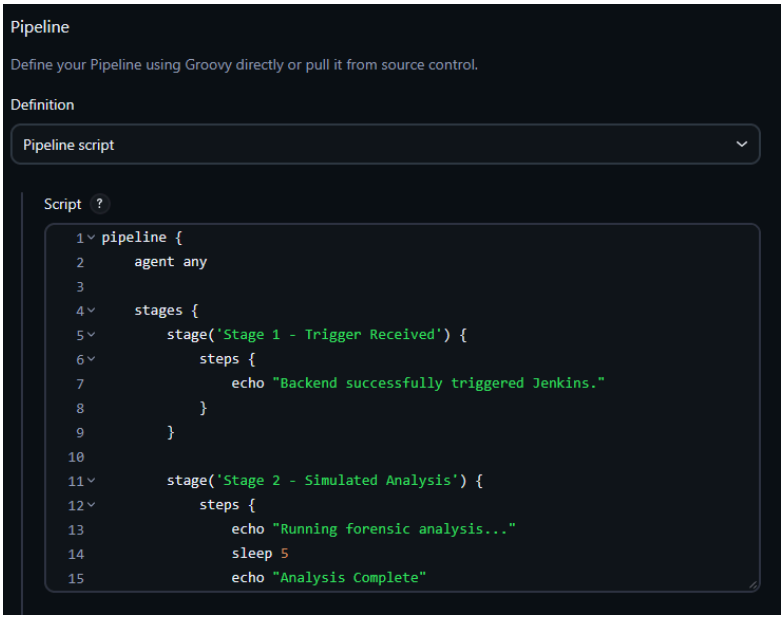
Trigger Received:
Confirms backend successfully initiated pipeline.

Simulated Analysis:
Represents execution of forensic processing logic.

Report Generation:
Simulates structured output generation.

Execution Model

Pipeline follows sequential execution model.
Each stage must complete before next begins.
Failure in any stage terminates pipeline.



5.4 Analysis Engine (Simulated)

Purpose

Represents the forensic analysis logic layer.

Responsibilities

- Execute simulated processing logic.

- Represent future integration with forensic tools (e.g., Volatility).
- Simulate time-based execution delays.

Future implementation may include:

- Memory dump parsing.
- Process list extraction.
- Malware detection logic.
- Network artifact analysis.

5.4 Logging and Monitoring Module

Jenkins provides console logs for each pipeline stage.

Backend logs HTTP request results.

System monitoring currently limited to console output but may be extended with database logging.

6. Data Design

6.1 Conceptual Data Model

The conceptual model consists of three entities:

User
AnalysisRequest
PipelineExecution

Relationships:

- A User can initiate multiple AnalysisRequests.
- Each AnalysisRequest triggers one PipelineExecution.

6.2 Logical Data Schema

Example schema design:

Users Table:

- user_id (Primary Key)
- username
- password_hash

AnalysisRequest Table:

- request_id (Primary Key)
- user_id (Foreign Key)
- status
- timestamp

PipelineExecution Table:

- execution_id
- request_id
- stage_name
- execution_time
- status

6.3 Data Storage Considerations

Although database storage is minimal in current implementation, the design allows:

- Storing forensic reports
- Storing execution logs
- Storing analysis metadata
- Indexes should be created on primary and foreign keys for faster lookup.

7. Interface Design

7.1 REST API Interface

Endpoint: POST /start-analysis

Authentication: None for frontend.
Backend uses Jenkins API token for secure access.

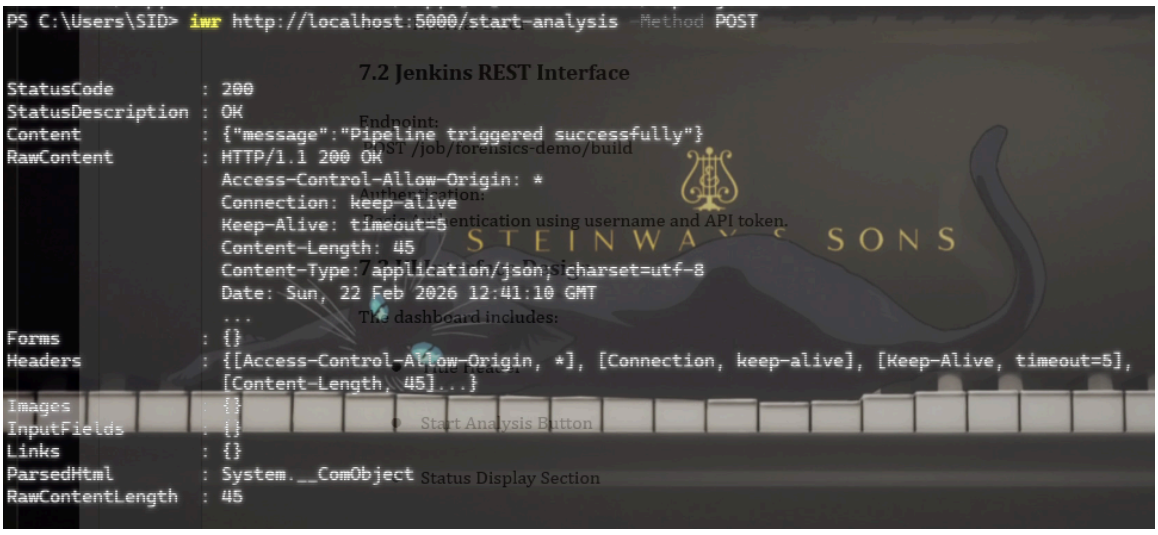
Response Codes:

200 – Success
500 – Internal Error

7.2 Jenkins REST Interface

Endpoint:
POST /job/forensics-demo/build

Authentication:
Basic Authentication using username and API token.



7.3 UI Interface Design

The dashboard includes:

- Title Header
- Start Analysis Button
- Status Display Section

Navigation Flow:

User → Dashboard → Click Button → Trigger Pipeline

8. Detailed Design for Major Algorithms / Rules

8.1 Backend Trigger Algorithm

Pseudo-code:

```
Receive HTTP request
 Authenticate with Jenkins
 Send POST request to Jenkins build endpoint
 If response is successful
 Return success message
 Else
 Return error
```

8.2 Pipeline Execution Logic

Sequential Workflow:

```
Stage 1 executes
 If successful → Stage 2 executes
 If successful → Stage 3 executes
 Else → Pipeline fails
```

9. Non-Functional / Quality Requirements Design

9.1 Performance

Pipeline execution designed to complete within minimal delay for demonstration.

9.2 Scalability

System architecture allows:

- Horizontal scaling of backend
- Distributed Jenkins agents
- Future microservice separation

9.3 Reliability

- Jenkins ensures stage-based monitoring and failure isolation.

9.4 Maintainability

Layered design ensures independent modification of frontend, backend, and pipeline logic.

10. Security Design

10.1 Authentication

Jenkins access secured using API token.

10.2 Authorization

Only authenticated backend can trigger pipeline.

10.3 Future Enhancements

- JWT-based authentication
- HTTPS implementation
- Role-based access control

11. Deployment & Operations

11.1 Environment Setup

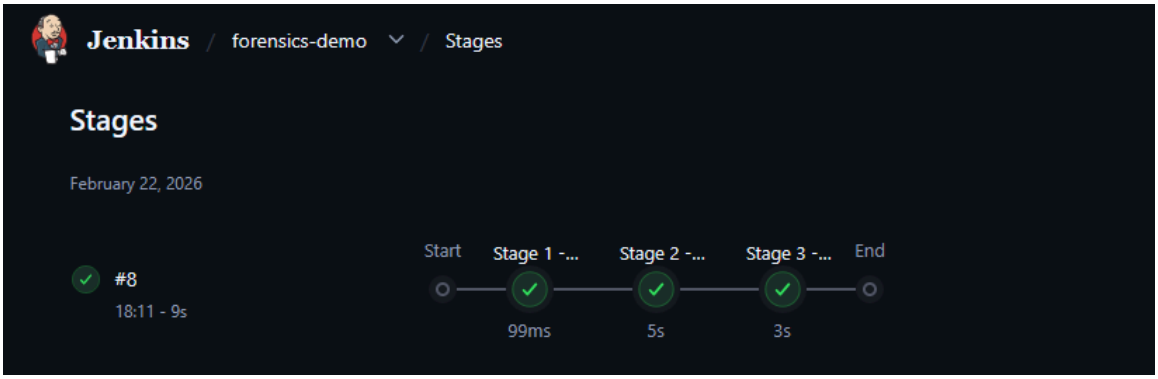
- Install Node.js
- Install Docker

- Run Jenkins container
- Start backend server
- Start frontend server

```
PS C:\Users\SID\Documents\memory-forensics\memory-forensics-backend> node server.js
Backend running on http://localhost:5000
```

11.2 Operational Workflow

- Start services.
- Access dashboard.
- Trigger analysis.
- Monitor Jenkins pipeline.
- Review execution logs.



12. Testing Strategy

12.1 Unit Testing

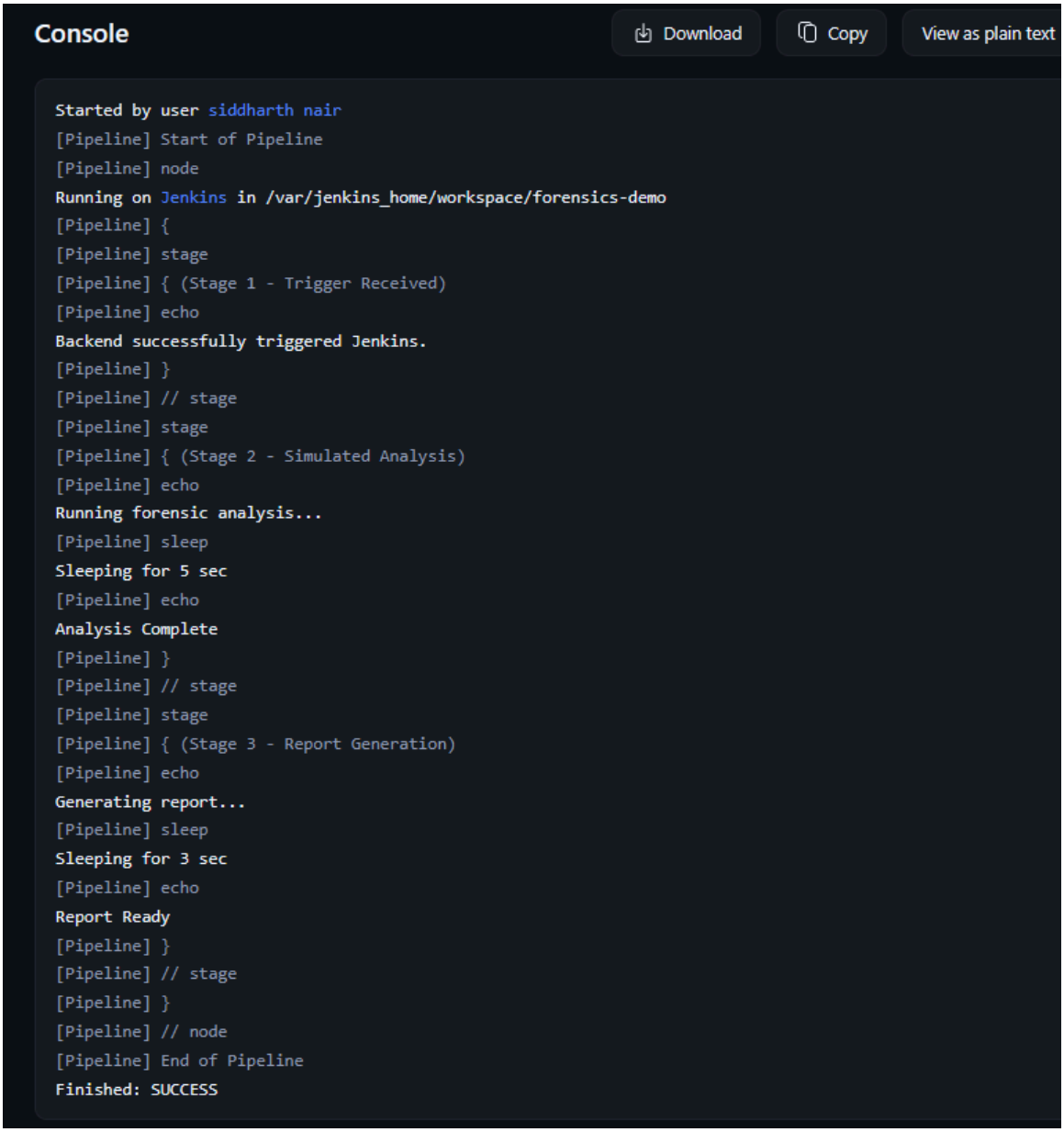
Backend endpoint validation.

12.2 Integration Testing

Verify backend successfully triggers Jenkins pipeline.

12.3 System Testing

Full end-to-end execution validation.



13. Conclusion

This Software Design Document provides a detailed architectural and technical description of the Memory Forensics Automation System. The system integrates a MERN stack web application with a Jenkins CI pipeline to automate forensic workflow orchestration.

The design ensures:

- Modular architecture
- Clear separation of concerns
- Secure REST-based communication
- CI/CD integration
- Extensibility for future forensic enhancements

The document establishes a strong foundation for further development, deployment, and scaling of the system.