SIEM Application System Architecture

1. Overview

The SIEM (Security Information and Event Management) application is built using a modern, scalable architecture that leverages Django for the backend, React with Material UI for the frontend, and incorporates various components for efficient data processing, storage, and analysis.

2. High-Level Architecture

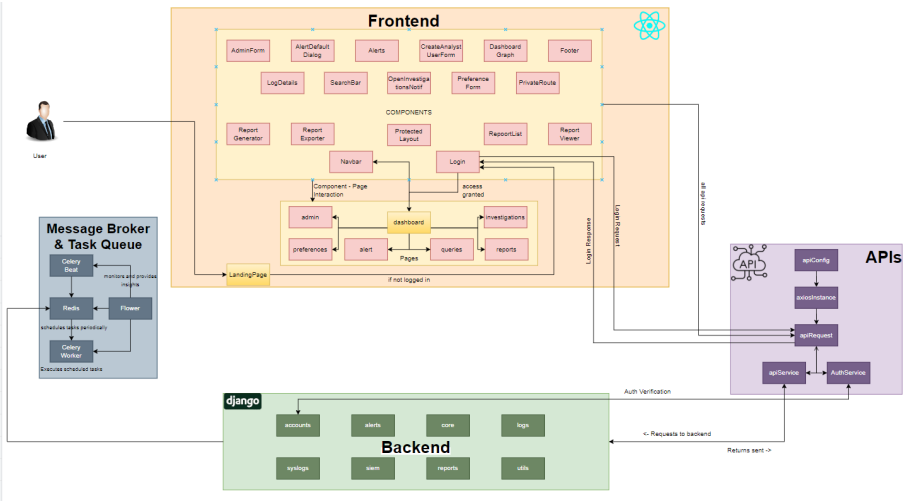

Fig:1 Architecture Diagram

3. Component Breakdown

3.1 Frontend

- **Technology**: React with Material UI
- **Key Features**:
  - Single Page Application (SPA) architecture
  - Responsive design for various device sizes
  - Component-based structure for reusability

3.2 Backend

- **Framework**: Django (Python-based)
- **Key Components**:
  - **Django REST Framework**: For building RESTful APIs
  - **SimpleJWT**: For JWT-based authentication
  - **Corsheaders**: For handling Cross-Origin Resource Sharing
  - **Django Filters**: For advanced query filtering

  - **DRF Spectacular**: For API documentation

3.3 Database

- **Technology**: MySQL
- **Key Aspects**:
  - Stores user data, alerts, logs, and other application-specific information
  - Configured with environment variables for flexibility

3.4 Caching and Message Broker

- **Technology**: Redis
- **Uses**:
  - Celery broker for task queue management
  - Potential use for caching frequently accessed data

3.5 Task Queue

- **Technology**: Celery
- **Key Tasks**:
  - Periodic log checking and processing
  - Background processing of time-consuming operations

3.6 Logging

- **Implementation**:
  - Custom logging configuration
  - File-based logging with rotating log files
  - Console logging for development

3.7 Authentication and Authorization

- **Method**: JWT (JSON Web Tokens)
- **Features**:
  - Token-based authentication for API access
  - Role-based access control (RBAC) using custom User and Role models

4. Data Flow

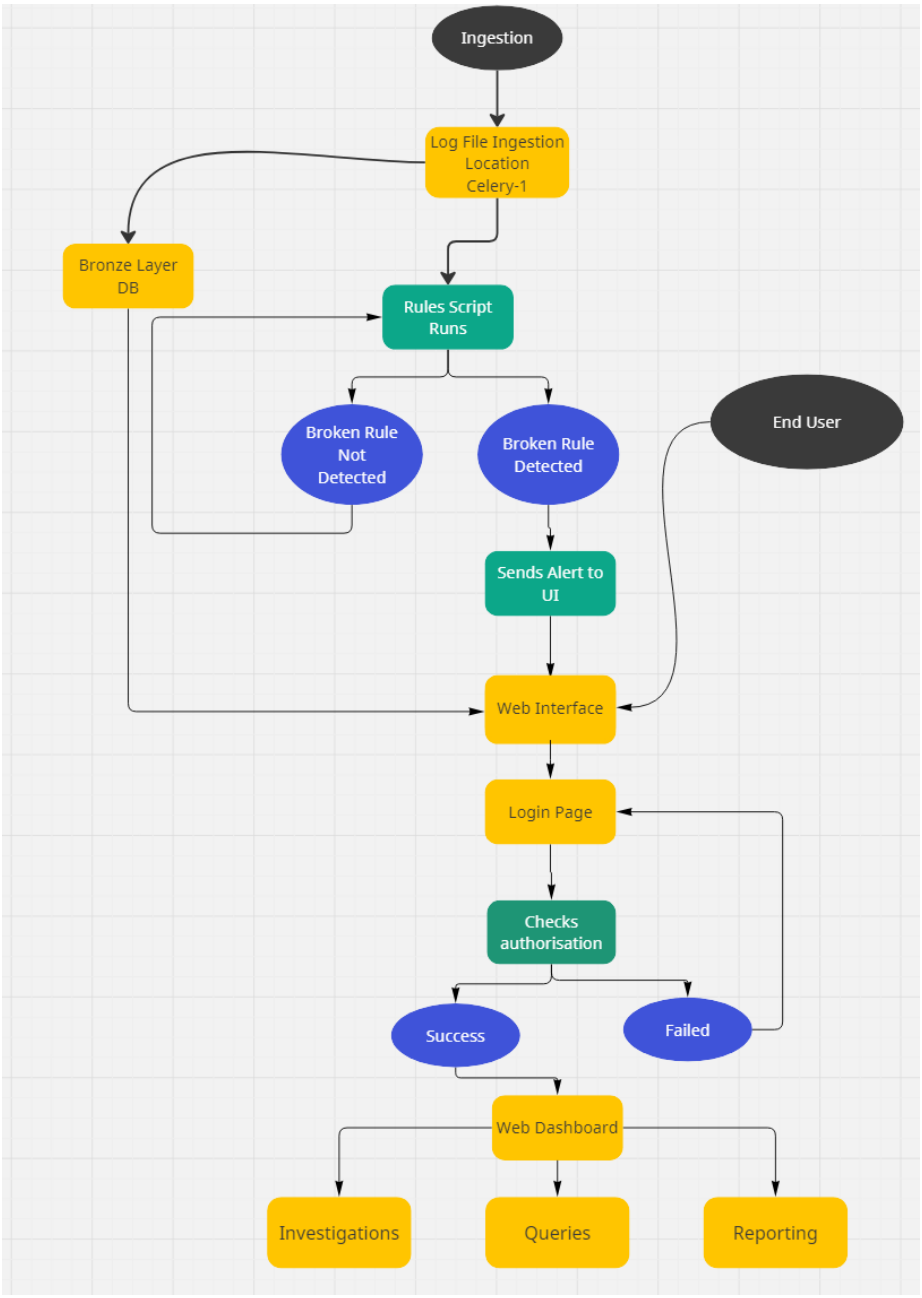

Fig:2 Data Flow Diagram

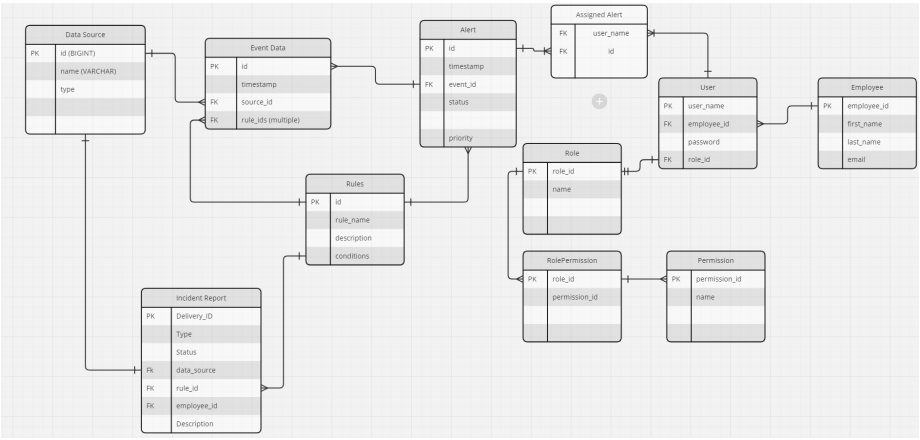

Fig:3 ERD database Diagram

1. **Log Ingestion**:
    - External systems write logs to a designated directory
    - Celery task (`check_and_process_logs`) periodically checks for new logs
    - New logs are processed and stored in the `BronzeEventData` model
2. **Alert Generation**:
    - Processed log data is analyzed against defined rules
    - Matching events trigger the creation of `Alert` objects
3. **User Interaction**:
    - Users interact with the React frontend
    - API requests are sent to Django backend
    - Django processes requests, interacting with the database as needed
    - Responses are sent back to the frontend for display
4. **Reporting**:
    - Users can generate reports based on alerts and log data
    - Report generation may be handled as a background task for large datasets

5. API Structure

- **Version**: v1
- **Base URL**: `/api/v1/`
- **Key Endpoints**:
    - `/accounts/`: User management
    - `/alerts/`: Alert management
    - `/logs/`: Log data access
    - `/reports/`: Report generation and management

6. Security Measures

- HTTPS for all client-server communication
- JWT for API authentication
- CORS configuration to restrict allowed origins
- API rate limiting to prevent abuse
- Secure password hashing using Django's built-in mechanisms
- Traefik handles SSL/TLS, ensuring encrypted communication
- API authentication bypass should be disabled in production
- Database credentials are managed via environment variables
- Traefik's API is exposed insecurely (–api.insecure=true) and should be secured in production

7. Scalability Considerations

- Stateless application design for horizontal scaling
- Use of Celery for offloading time-consuming tasks
- Potential for database read replicas for scaling read operations
- Containerization-ready for easy deployment and scaling
- The containerized architecture allows for easy horizontal scaling of frontend and backend services
- Traefik can load balance between multiple instances of services
- Database scaling would require additional setup for replication or clustering

8. Monitoring and Maintenance

- Comprehensive logging for application events and errors
- Celery task monitoring for background job performance
- Database query monitoring for performance optimization
- Traefik is configured with debug-level logging
- Application logs are likely captured by the Docker logging driver
- Consider integrating with a centralized logging solution for production

9. Development and Deployment

9.1 Container Orchestration

The application is containerized using Docker, with services defined in a docker-compose file for easy deployment and scaling. The key components in the production environment are:

1. **Frontend**
    - Built from a custom Dockerfile in the `./frontend` directory

- Exposed via Traefik reverse proxy
- Environment variables:
  - `REACT_APP_API_URL`: URL for backend API
  - `CHOKIDAR_USEPOLLING`: Enabled for hot reloading
  - `REACT_APP_BYPASS_AUTH`: Authentication bypass flag (should be disabled in production)
  - `DOMAIN`: Domain name for the application

2. **Backend**
   - Built from a custom Dockerfile in the `./backend` directory
   - Runs on port 8000 internally, mapped to a configurable external port
   - Also listens on UDP port 514 (likely for syslog ingestion)
   - Uses environment variables for configuration
3. **Database**
   - Uses MariaDB 10.5
   - Data persisted in a named volume
   - Initializes with a custom SQL script
   - Exposed on port 3306 internally, mapped to a configurable external port
4. **Traefik**
   - Acts as a reverse proxy and load balancer
   - Handles SSL termination using Let's Encrypt
   - Configured for Cloudflare DNS challenge
   - Exposes dashboard on port 8080

9.2 Networking

- Traefik handles routing and SSL termination
- Frontend is accessible via HTTPS on the configured domain
- Backend API is accessible internally, routed through Traefik
- Database is accessible within the Docker network and optionally exposed externally

9.3 SSL/TLS

- SSL certificates are automatically provisioned and renewed using Let's Encrypt
- Cloudflare is used for DNS challenges, requiring Cloudflare API credentials

9.4 Persistence

- MariaDB data is persisted using a named Docker volume
- Log files and processed log files are stored in named volumes
- Traefik SSL certificates are stored in a bind-mounted directory

9.5 Environment Configuration

- Extensive use of environment variables for configuration
- Sensitive information (API keys, passwords) should be managed securely, potentially using Docker secrets in a production swarm

10. Continuous Integration and Continuous Deployment (CI/CD)

The SIEM application utilizes GitHub Actions for automated testing, building, and deployment, ensuring code quality and streamlining the deployment process.

10.1 Continuous Integration (CI)

The CI pipeline is triggered on pushes to `main` and `develop` branches, as well as on pull requests to these branches.

Key steps in the CI process:

1. **Environment Setup**:
    - Uses Ubuntu latest as the runner
    - Sets up a MariaDB service container for testing
2. **Backend Testing**:
    - Sets up Python 3.12
    - Installs backend dependencies
    - Runs Django tests against the MariaDB test database
3. **Frontend Testing and Building**:
    - Sets up Node.js 20.18.0
    - Installs frontend dependencies
    - Runs React tests
    - Performs ESLint checks
    - Builds the frontend application

This CI process ensures that all tests pass and the application builds successfully before any code is merged or deployed.

10.2 Continuous Deployment (CD)

The CD pipeline is triggered on pushes to the `main` branch, automating the deployment process to the production environment.

Key steps in the CD process:

1. **Docker Image Building and Pushing**:
    - Sets up Docker Buildx
    - Logs in to Docker Hub using secrets

- Builds Docker images using `docker-compose.prod.yml`
- Pushes built images to Docker Hub

2. **Deployment to Production Server**:
   - Connects to the production server via SSH
   - Pulls the latest code from the main branch
   - Pulls the latest Docker images
   - Starts or restarts services using Docker Compose
   - Prunes old Docker resources

10.3 Security Considerations in CI/CD

- Sensitive information (Docker Hub credentials, SSH keys) is stored as GitHub secrets
- The production server's SSH key is used for secure deployment
- Docker Hub is used as a secure registry for Docker images

10.4 Workflow Benefits

- **Automated Testing**: Ensures code quality and catches issues early
- **Consistent Builds**: Guarantees that the application builds successfully in a clean environment
- **Streamlined Deployment**: Automates the process of updating the production environment
- **Version Control**: Keeps production in sync with the main branch
- **Rollback Capability**: Easy to roll back to previous versions if issues are detected

10.5 Areas for Improvement

- Implement staging environment deployment for testing before production
- Add automated security scanning of Docker images
- Implement database migration steps in the deployment process
- Consider blue-green deployment strategy for zero-downtime updates

This CI/CD setup provides a robust pipeline for testing, building, and deploying your SIEM application, ensuring that only tested and approved code reaches the production environment.

11. Development Workflow

With the CI/CD pipeline in place, the development workflow typically follows these steps:

1. Developers work on feature branches
2. Pull requests are created to merge into `develop` or `main`
3. CI runs on the pull request, checking tests and build
4. Code review is performed

5. Once approved and CI passes, code is merged
6. On merge to `main`, CD pipeline deploys to production

This workflow ensures code quality, facilitates collaboration, and automates the deployment process, allowing for rapid and reliable updates to the SIEM system.

12. Future Enhancements

- Implementation of real-time alerts using WebSockets
- Integration with external threat intelligence feeds
- Advanced analytics and machine learning for anomaly detection
- Expansion of reporting capabilities with interactive dashboards

This architecture provides a solid foundation for a scalable and maintainable SIEM system, with room for future growth and enhancements as the application's needs evolve.