**Augury**

Design Document

**Authored by:**

Kaila Bryant

Ian Chorne

Nahod Habte

Liam Ison

Garrett McGuire

Date: (02/23/2025)

Department: College of Computing and Software Engineering

Course: SWE 4724 Capstone Project

Professor: Yan Huan

Table of Contents

[1. INTRODUCTION 3](#_Toc190782504)

[1.1 Purpose 3](#_Toc190782505)

[1.2 Scope 3](#_Toc190782506)

[1.3 Overview 3](#_Toc190782507)

[1.4 Definitions and Acronyms 3](#_Toc190782508)

[2. SYSTEM OVERVIEW 3](#_Toc190782509)

[3. SYSTEM ARCHITECTURE 3](#_Toc190782510)

[3.1 Architectural Design 3](#_Toc190782511)

[3.2 Technical design 5](#_Toc190782512)

[3.3 Design Rationale 6](#_Toc190782513)

[4. DATA (DATABASE) DESIGN 6](#_Toc190782514)

[4.1 Overview 6](#_Toc190782515)

[4.2 Entities and Relationships 6](#_Toc190782516)

[4.2.1 ERD 6](#_Toc190782517)

[4.3 External Dependencies 7](#_Toc190782518)

[4.4 Hosting PostgreSQL server for Development 7](#_Toc190782519)

[5. HUMAN INTERFACE DESIGN 7](#_Toc190782520)

[5.1 UI design 7](#_Toc190782521)

[5.2 UX design 8](#_Toc190782522)

[User Flow: 8](#_Toc190782523)

### INTRODUCTION

## Purpose

This software design document describes the architecture and system design of Augury, a security lookup interface. Through the various sections in this document, we explain the technology that will be used to build Augury and our reasoning behind it.

## Scope

Our application will assist cyber security analysts at Cox Enterprises quickly and efficiently look up potential IOCs and save the results of their analysis. The benefits of the program will be improved organization, visual appeal, and speed over the current application for COX’s analysts.

## Overview

This document will explain the design choices and rationale for the application. Further sections of the document will go into greater detail about the system architecture, detailed design, database design, human interface design, and requirements matrix.

## Definitions and Acronyms

Acronyms and definitions for this document are as follows:

**IOC** – Indicator of Compromise

**MVC** – Model View Controller

**ERD**- Entity Relationship Diagram

**IDE** – Integrated Development Environment

### SYSTEM OVERVIEW

Our application is a lookup interface that allows cyber security analysts to quickly analyze potential indicators of compromise. It is a web-based application that runs locally for security and ease of access. Users need to be able to use the tool quickly and easily, so it needs to be kept lightweight and readable, so some features like user profiles are jettisoned to allow for simplicity and ease of access.

### SYSTEM ARCHITECTURE

## Architectural Design

Augury uses the Model View Controller (MVC) software architecture pattern. We selected this pattern due to our developers’ familiarity with it, and for its simplicity. Systems using this pattern have three main types of components:

* Views – The user interface and web pages the end users will interact with. These accept user input which is then passed through to the controllers in the back end.
* Controllers – These back-end scripts contain functions that will receive user requests from the views and send the request to the proper end point with a model.
* Models – These represent actual data that can be modified after receiving requests from the controllers. After the request is completed, the results are sent back to the controller, which then sends it back to the view.

A diagram of a controller

AI-generated content may be incorrect.

Figure 1: Architectural Design

Our system's efficiency is guaranteed by this structured data flow, which keeps the user interface, logic processing, and data management distinct. The MVC pattern is used by Augury to gain scalability, modularity, and to improve user experience.

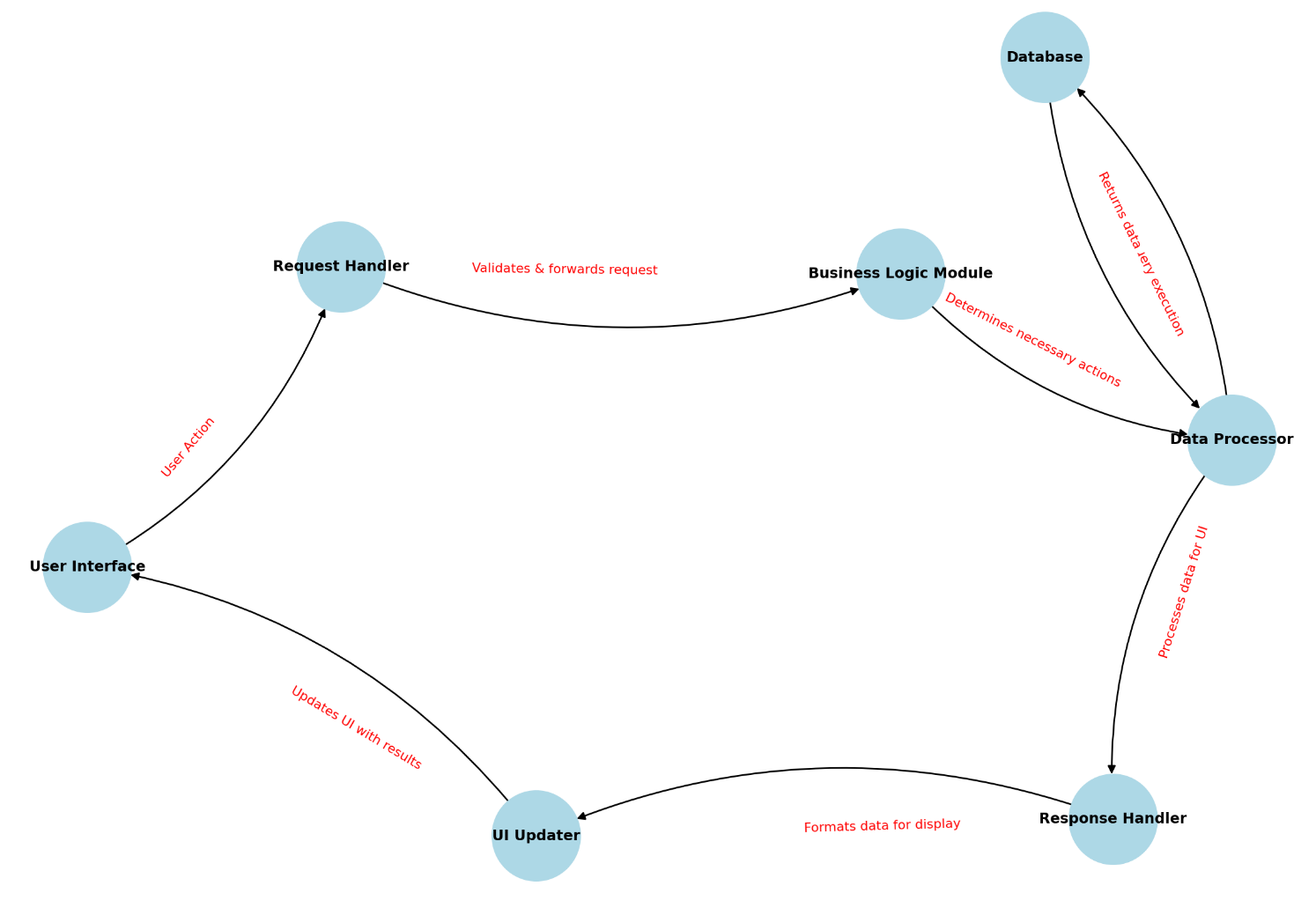


Figure 2 - Data Flow Diagram

The primary interface through which users submit queries and receive displayed results is the User Interface (View Layer). When a user starts an action, such as searching for data, the request is routed to the Controller Layer's Request Handler. By serving as a point of contact, this component makes sure that requests are formatted correctly and routed to the relevant business logic.

The request is sent to the Data Processor in the Model Layer after being further validated and the necessary actions determined by the Business Logic module. The database, which acts as the persistent storage system, communicates with the data processor to guarantee that data updates and retrieval are carried out effectively.

Once the database returns the requested data, it flows back through the Data Processor, where it is structured into a usable format. The Response Handler then processes this structured data, ensuring it is formatted appropriately for display. Finally, the UI Updater modifies the User Interface, presenting the final processed information to the user.

For Augury, this flow will take its form via our front-end UI scripts (views), which send user input to our back-end logic and the Count FAKEula API (controllers) which interacts with the lookup endpoints and database (models) as needed.

## Technical design

Augury will be developed using the Go programming language for the front and back end of the application. These scripts will interact with the Flask API provided by cox to communicate with the database, and security lookup endpoints within the API.

To develop the system, we will be using the VSCode IDE due to its simplicity, speed, and versatility. Each of our developers are familiar with it and its various tools.

## Design Rationale

MVC architecture was selected due to our developers’ familiarity of it, and due to its purpose as a web application design pattern. Web applications naturally fit within this type of architecture due to the number of pages users will interact with, the back-end functions that receive this input, and the data that the users want to access and modify. Each critical component of a web application has a designated spot within the design pattern; thus, it seemed a natural choice.

### DATA (DATABASE) DESIGN

## 4.1 Overview

The purpose of this database is to store metadata about IOC (Indicator of Compromise) lookups performed by users. The system does not store actual IOC details or threat intelligence data, as that information is retrieved dynamically from external threat intelligence APIs. This database will only track:

* When and IOC was queried
* What IOC was queried
* How many results are returned from external API’s

## 4.2 Entities and Relationships

The database consists of a single QueryLogs table to record lookup events. External entities IOC API and userAPI represent systems that provide threat intelligence and authentication services.

## 4.2.1 ERD

Augury’s entity relationship diagram is shown below:

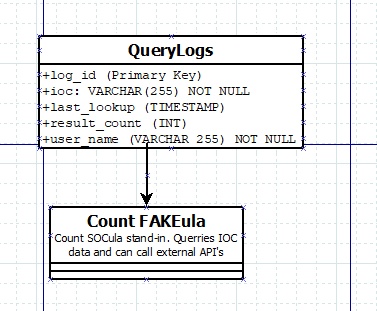


Figure 3-Augury ERD

4.2.2 Table Schema

Augury’s table scheme is shown below:

## 

Figure 4-Augury Table Schema

## 4.3 External Dependencies

* **IOC Lookup APIs:** The application queries Count FAKEula, which serves as a dummy API for testing the internal tool.

## 4.4 Hosting PostgreSQL server for Development

For development purposes, the PostgreSQL database will be hosted using **Heroku PostgreSQL**, a cloud-based relational database service.

This setup allows for:

* **Easy setup for development:** Heroku offers a simple deployment process at a free tier that fits with our requirements. This is ideal for development and testing.
* **Simple Migration for deployment:** Once the system is ready for deployment the database can easily be migrated to any enterprise solution like AWS RDS, or Azure servers.

### HUMAN INTERFACE DESIGN

## UI design

Augury's User Interface Design prioritizes clarity, efficiency, and ease of use, allowing COX cybersecurity analysts to quickly look up and analyze IOCs. Usually analysts work under time constraints, so the UI will be designed for speed and simplicity while maintaining a professional, modern style.

* The interface will avoid unnecessary visual clutter to allow users to focus on IOC results.
* Centered Input Box: Clearly labeled for entering IOCs.
* Minimal UI Elements: Removes distractions so users can focus on query entry.
* Dark Background with High-Contrast Text: Ensures readability and accessibility.

Key UI Features:

* Tabbed Results View: Users can open multiple IOC lookups in different tabs.
* Sortable & Filterable Data: Analysts can sort columns or apply filters to refine results.
* Persistent Input Field: Allows users to modify or re-query an IOC without leaving the page.

Key UI Enhancements:

* Multiple Tabs: Users can manage multiple IOC searches within the same session.
* Users can search within the results for specific IOC patterns.
* The export button allows analysts to save results for later reference or reporting.
* Interaction Design:
* Users can toggle filters to customize their view.
* The interface supports real-time text search, navigation buttons, and highlighting of search results.

## UX design

Security analysts at COX sometimes receive communication regarding an IOC and often need to analyze it quickly to mitigate the risk of a cyber-attack. Due to this, Augury will be simple, responsive, and fast. Users will be able to read information quickly, navigate through the web pages quickly, and analyze IOCs in an efficient manner.

## 

## User Flow:

Receiving an IOC Notification

* Analysts copy an IOC from their security feed.

Entering the IOC in Augury:

* The home page includes a clean input field with few distractions.
* Pressing the "Search" button immediately processes the request.

Reviewing Results

* The IOC intelligence is presented in an easy-to-read table format.
* Users can toggle filters to modify their analysis.
* New searches open in separate tabs without losing previous results.

Finding Specific Details

* The built-in "Find" function enables analysts to locate specific matches within a large dataset.
* They can use dropdown filters to narrow down search types (such as IPs and domains).

Exporting Results

* Analysts can export their results for reporting and documentation.