FORM NO. F/ TL / 024
Rev 00 Date 20.03.2020

# REAL-TIME PHISHING DETECTION WITH MACHINE LEARNING INTEGRATION

(PROJECT REPORT PHASE- II)

*submitted in partial fulfillment of the requirements*

*for the award of the degree in*

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

by

**Vetrivel M (211061101494)**

**Vishnu sah V T (211061101505)**

**Tarun Vikash B (211061101831)**

Dr. M.G.R.
EDUCATIONAL AND RESEARCH INSTITUTE
DEEMED TO BE UNIVERSITY
University with Graded Autonomy Status
(An ISO 21001 : 2018 Certified Institution)
Periyar E.V.R. High Road, Maduravoyal, Chennai-95. Tamilnadu, India.

**DEPARTMENT**

**OF**

**COMPUTER SCIENCE AND ENGINEERING**

**APRIL 2025**

FORM NO. F/ TL / 024
Rev 00 Date 20.03.2020

Dr. M.G.R.
EDUCATIONAL AND RESEARCH INSTITUTE
DEEMED TO BE UNIVERSITY
University with Graded Autonomy Status
(An ISO 21001 : 2018 Certified Institution)
Periyar E.V.R. High Road, Maduravoyal, Chennai-95. Tamilnadu, India.

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## BONAFIDE CERTIFICATE

This is to certify that this Project Report (Project Phase-I) is the bonafide work of **Mr. VETRIVEL M** Reg. No **211061101494**, **Mr. VISHNU SAH VT** Reg. No **211061101505**, Mr. **TARUN VIKASH B** Reg. No **211061101831**, who carried out the project entitled "**REAL-TIME PHISHING DETECTION WITH MACHINE LEARNING INTEGRATION**" under our supervision from December 2024 to May 2025.

| Internal Guide | Project Coordinators | Department Head |
|---|---|---|
| **Mrs.S.DIVYA**<br>Assistant Professor-CSE<br>Dr. M.G.R Educational and Research Institute<br>Deemed to be University | **Dr.G.SONIYA PRIYATHARSINI**<br>Professor-CSE<br>**Mr.G.SENTHILVELAN**<br>Assistant Professor-CSE<br>Dr. M.G.R Educational and Research Institute<br>Deemed to be University | **Dr.S.GEETHA**<br>Professor and HoD of CSE<br>Dr. M.G.R Educational and Research Institute<br>Deemed to be University |

**Submitted for Viva Voce Examination held on** ________________

**Internal Examiner** **External Examiner**

# DECLARATION

We, **VETRIVEL M (211061101494), VISHNU SAH V T(211061101504), TARUN VIKASH B (211061101831)**, hereby declare that the Project Report (Project Phase-I) entitled "**REAL-TIME PHISHING DETECTION WITH MACHINE LEARNING INTEGRATION**" is done by us under the guidance of **Mrs.S.DIVYA** is submitted in partial fulfillment of the requirements for the award of the degree in BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING.

1.

2.

3.

**DATE:**

**PLACE: CHENNAI** **SIGNATURE OF THE CANDIDATE(S)**

## ACKNOWLEDGEMENT

We would first like to thank our beloved Chancellor **Thiru.Dr. A.C. SHANMUGAM, B.A., B.L.,** President **Er. A.C.S. Arunkumar, B.Tech., M.B.A.,** and Secretary **Thiru A. RAVIKUMAR** for all the encouragement and support extended to us during the tenure of this project and also our years of studies in this wonderful University.

We express my heartfelt thanks to our Vice Chancellor **Prof. Dr. S. GEETHALAKSHMI** in providing all the support of our Project (Project Phase-I).

We express my heartfelt thanks to our Head of the Department, **Prof. Dr. S.Geetha**, who has been actively involved and very influential from the start till the completion of our project.

Our sincere thanks to our Project Coordinators **Dr.G.SONIYA PRIYATHARSINI** and **Mr.G.SENTHILVELAN** and Project guide **Mrs.S.DIVYA** for their continuous guidance and encouragement throughout this work.

We would also like to thank all the teaching and non-teaching staffs of Computer Science and Engineering department, for their constant support and the encouragement given to us while we went about to achieving our project goals.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| **UI** | User Interface |
| **HTML** | Hyper Text Markup Language |
| **CSS** | Cascading Style Sheets |
| **AI** | Artificial Intelligence |
| **ML** | Machine Learning |
| **URL** | Unified Resource Locator |
| **REST** | REpresentational State Transfer |
| **API** | Application Programming Interface |
| **JSON** | JavaScript Object Notation |
| **RBAC** | Role-Based Access Control |
| **PostgreSQL** | Post-Ingres Structured Query Language |
| **Oauth** | Open Authorization |
| **NLP** | Natural Language Processing |

## ABSTRACT

Our work introduces a real-time phishing detection system that leverages machine learning models to accurately detect and prevent phishing attacks, spam, and defaced websites. The system evaluates multiple key URL attributes, including length, domain, subdomain, numeric values, special characters, and HTTPS presence, ensuring a thorough approach to phishing identification. By utilizing machine learning classifiers such as Decision Tree, Random Forest, and Extra Trees, the system achieves high precision in differentiating between legitimate and harmful websites. Furthermore, it incorporates behavioral analysis and continuous real-time monitoring to improve detection efficiency, making it adaptable to new phishing strategies. With its ability to provide instant threat alerts, this system enhances user security and offers robust protection against cyber risks in online environments

**Keywords:** Artificial Intelligence (AI), cybersecurity, Threat Detection, Threat Response, Machine Learning, Paradigm Shift, Evolution

| | | | |
|---|---|---|---|
| Student Group | VISHNU SAH V T (211061101505) | VETRIVEL M (211061101494) | TARUN VIKASH B (211061101831) |
| Project Title | REAL-TIME PHISHING DETECTION WITH MACHINE LEARNING INTEGRATION | | |
| Program Concentration Area | Phishing, Spamming, hacking | | |
| Constraints Example | Power Constraints | | |
| Economic | No | | |
| Environmental | No | | |
| Sustainability | Yes | | |
| Implementable | Yes | | |
| Ethical | Yes | | |
| Health and Safety | No, yes | | |
| Social | Yes | | |
| Political | No | | |
| Other | Machine Learning Integration | | |
| Standards | | | |
| 1 | PEP 8 | | |
| 2 | ISO 50001, ISO 27001, ISO 12207, ISO 27001 | | |
| 3 | USB to TTL | | |
| Prerequisite Courses for the Major Design Experiences | 1. Networking<br>2. Machine Learning<br>3. Cyber Security | | |

# CHAPTER-1

# INTRODUCTION

## 1.1.Introduction:

In today's interconnected world, the internet has become an integral part of daily life for billions of people. As more activities shift online, from personal communications to banking and e-commerce, the risk of cyber threats continues to rise. Among these, phishing attacks have emerged as one of the most common and dangerous forms of cybercrime.

Phishing refers to the fraudulent practice of sending fake messages or creating deceptive websites that mimic legitimate ones to lure users into disclosing sensitive information, such as usernames, passwords, and credit card details. These attacks not only compromise individuals' privacy and finances but also threaten the security of organizations and societies at large.

The sophistication and frequency of phishing attacks have increased exponentially in recent years, driven by the widespread adoption of digital platforms and the availability of phishing toolkits. Traditional security mechanisms such as firewalls and antivirus software, while essential, are often inadequate against phishing.

Unlike viruses or malware, phishing primarily exploits human trust and can evade technical defenses by using socially engineered tactics. Therefore, there is a critical need for proactive, intelligent systems that can identify and block phishing attempts before they reach unsuspecting users.

To address this need, researchers and cybersecurity professionals have increasingly turned to Machine Learning (ML) as a potential solution. ML models can analyze large datasets, recognize patterns, and make predictions based on empirical data. In the context of phishing detection, ML models can be trained to differentiate between legitimate and malicious websites or emails based on features such as URL structure, domain characteristics, and the presence of certain keywords or characters. This approach enables the development of a Real-Time Phishing Detection System that can identify phishing attempts with high accuracy, often before any harm is done.

## 1.2.Problem Statement

In an ideal world, people would easily recognize and avoid malicious URLs designed to steal their data. However, social engineering attacks have evolved, exploiting users' trust through deceptive links, ultimately leading to data theft and system compromise. Our system addresses this issue by analyzing URLs in real time and alerting users when they encounter a phishing or spam link and even gathering all the link present in a web page, this helping them avoid dangerous websites and protect their information. In reality, attackers use increasingly sophisticated methods beyond URLs — like distributing USB drives or OTG cables embedded with hidden executable malware.

Once connected to a device, these malicious tools can steal data almost instantly, especially when the system is connected to the internet. Despite the variety of attack methods, phishing via fake URLs remains one of the most common entry points for cyberattacks. Our work specifically focuses on real-time URL analysis because phishing attacks through deceptive links remain a primary threat vector. By detecting and flagging suspicious URLs, the system aims to reduce the risk of data exploitation, providing users with immediate alerts to safeguard their personal and professional information.

## 1.3.Objectives of the Work

The primary objective of this work is to develop a Real-Time Phishing Detection System that leverages Machine Learning to accurately classify URLs as either "Phishing" or "Benign.".

The system is designed to offer a reliable, proactive solution capable of identifying phishing attempts based on the characteristics of a given URL, without reliance on outdated blacklists or predefined rules. Specific objectives include:

**Real-Time Detection**:

The system must be able to process and classify URLs in real-time to provide immediate feedback to users.

**Feature Engineering**:

Extract relevant features from URLs to improve model accuracy, such as URL length, domain structure, presence of special characters, and subdomain characteristics.

**Machine Learning Integration**:

Utilize multiple Machine Learning algorithms, including Decision Tree, Random Forest, and Extra Trees classifiers, to enhance detection accuracy and robustness.

**Ensemble Decision Making**:

Implement an ensemble approach that aggregates predictions from multiple models to achieve a more reliable final decision.

**API Development**:

Develop a RESTful API using **FastAPI** to enable integration with other applications or platforms for real-time phishing detection.

## 1.4.Significance of the Work

The Real-Time Phishing Detection System with Machine Learning integration represents a significant advancement in cybersecurity. Traditional phishing detection techniques, such as blacklist-based approaches, cannot keep up with the rapidly changing landscape of phishing threats.

Cybercriminals frequently modify URLs, use temporary domains, and exploit new tactics to bypass these basic security measures. Machine Learning, however, offers a promising alternative by enabling the system to learn from data and adapt to new patterns without extensive manual intervention.

Moreover, the real-time nature of this system provides immediate benefits for both individual users and organizations. For individuals, it enhances protection against identity theft and financial fraud.

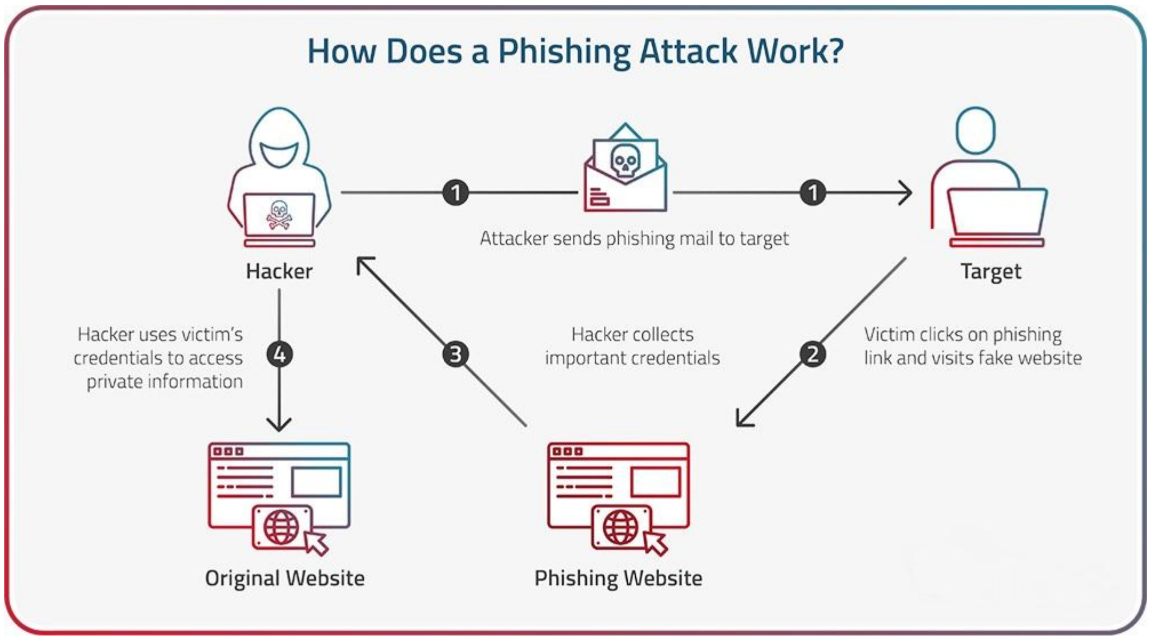

How Does a Phishing Attack Work?
Hacker
1
1
Attacker sends phishing mail to target
Target
Hacker uses victim's credentials to access private information
4
3
Hacker collects important credentials
2
Victim clicks on phishing link and visits fake website
Original Website
Phishing Website

Figure 1 - Actors involved in phishing

For organizations, it reduces the risk of data breaches, preserves reputation, and protects customer trust. Financial institutions, e-commerce platforms, and other sectors that handle sensitive data can particularly benefit from deploying such a system, as they are prime targets for phishing attacks.

From a technical standpoint, this work contributes to the development of intelligent phishing detection models that are both scalable and deployable. By incorporating an ensemble approach that aggregates predictions from multiple ML models, the system achieves a high level of accuracy and robustness.

# CHAPTER-2

# LITERATURE SURVEY

M. F. Ansari, P. K. Sharma, and B. Dash, “Prevention of phishing attacks using AI-based Cybersecurity Awareness Training,”

Ansari, Sharma, and Dash (2022), in their paper *“Prevention of Phishing Attacks Using AI-based Cybersecurity Awareness Training”*, examine the role of artificial intelligence in enhancing cybersecurity awareness to combat phishing threats. Published in the *International Journal of Smart Sensor and Adhoc Network*, this study discusses how AI-driven training programs can improve users' ability to recognize and respond to phishing attempts. By simulating phishing scenarios and providing targeted feedback, the AI-based training model helps users develop critical skills to identify phishing attacks, thus contributing to a more proactive and informed approach to cybersecurity.

W. Ali, “Phishing website detection based on supervised machine learning with wrapper features selection,”

Ali, explores the use of supervised machine learning models combined with wrapper-based feature selection to improve phishing website detection. Published in the *International Journal of Advanced Computer Science and Applications*, this study focuses on selecting the most relevant features from URLs, such as domain-specific characteristics, to enhance the accuracy and efficiency of machine learning classifiers in distinguishing phishing websites from legitimate ones. The findings demonstrate that applying wrapper feature selection methods can significantly improve model performance, making this approach effective for real-time phishing detection.

R. Alabdan, “future internet Phishing Attacks Survey: Types, Vectors, and Technical Approaches”, doi: 10.3390/fi12100168.

Alabdan (2020), in the paper *“Phishing Attacks Survey: Types, Vectors, and Technical Approaches”*, provides a comprehensive survey on the various forms of phishing attacks, their delivery methods, and the technical strategies used to combat them. Published in *Future Internet*, this paper categorizes phishing attacks based on their types and vectors, offering insights into the tactics used by attackers. Additionally, it examines the technical approaches employed in phishing detection and prevention, including advancements in AI and machine learning. Alabdan's work serves as a valuable resource for understanding the evolving nature of phishing threats and the effectiveness of modern defense mechanisms.

R. S. Rao, Alwyn, and R. Pais, "Detection of phishing websites using an efficient feature-based machine learning framework," Neural Computing and Applications, vol. 31, doi: 10.1007/s00521-017-3305 -0.

Rao, Alwyn, and Pais (2017), in their paper *"Detection of Phishing Websites Using an Efficient Feature-Based Machine Learning Framework"*, present a machine learning approach for detecting phishing websites based on carefully selected features. Published in *Neural Computing and Applications*, this study proposes an efficient framework that leverages URL-specific attributes—such as URL length, domain age, and presence of special characters—to improve classification accuracy. By utilizing feature-based techniques, the framework enhances the model's ability to distinguish phishing websites from legitimate ones effectively. This work highlights the importance of feature selection in improving both the performance and computational efficiency of phishing detection systems.

A. Kumar Jain and B. B. Gupta, "Towards detection of phishing websites on client-side using machine learning based approach," vol. 68, pp. 687–700, 2018, doi: 10.1007/s11235-017-0414-0.

Jain and Gupta (2018), in their paper *"Towards Detection of Phishing Websites on Client-Side Using Machine Learning Based Approach"*, explore a client-side solution for identifying phishing websites using machine learning techniques. Published in *Telecommunication Systems*, this study focuses on applying machine learning algorithms directly on the client-side, allowing for real-time detection without relying on server-based solutions. By analyzing features extracted from URLs, such as domain information and security indicators, the proposed approach enhances client-side security and protects users from phishing attacks at the point of interaction. This work contributes to the development of efficient, user-focused phishing detection tools that operate locally to ensure immediate protection.

# CHAPTER-3

# SYSTEM ANALYSIS

In order to develop a robust and effective real-time phishing detection system, a comprehensive analysis of the system requirements is essential. This phase ensures that all functional and non-functional aspects of the system are well-defined and understood, guaranteeing alignment with user needs and technical specifications.

## 3.1.Existing Systems

Several existing systems are designed to detect phishing websites and prevent users from falling victim to such attacks. These systems employ a variety of techniques, such as blacklists, heuristic-based detection, and machine learning. Some well-known existing systems include:

**Google Safe Browsing**:

Integrated into browsers like Chrome and Firefox, Google Safe Browsing uses an extensive blacklist of known phishing and malware websites. It warns users when they attempt to navigate to a malicious website. However, it primarily relies on previously identified phishing URLs, making it slower to respond to new phishing attempts.

**PhishTank**:

A community-driven system that allows users to submit suspected phishing URLs. These URLs are verified by the community and made available through an API. While effective for gathering phishing URLs, it is reliant on human verification and lacks real-time detection.

**Microsoft Defender SmartScreen**:

Built into Windows and Microsoft Edge, this system uses heuristic-based detection to warn users about suspicious websites. However, it can sometimes flag legitimate websites as phishing, resulting in a higher false positive rate.

**OpenPhish**:

Provides real-time phishing detection feeds, leveraging machine learning algorithms to detect phishing attacks. While it offers more advanced detection mechanisms, OpenPhish's full services are primarily subscription-based, limiting access for smaller developers or organizations.

## 3.2.Proposed System

The proposed system improves upon existing solutions by integrating machine learning models and behavioral analysis for real-time phishing detection. It leverages the strengths of multiple machine learning classifiers to ensure high accuracy in detecting both known and unknown phishing threats.

**Machine Learning Integration**:

Unlike traditional blacklist or heuristic-based systems, the proposed system uses advanced machine learning models to classify URLs based on features like URL length, domain structure, and special characters.This enables more adaptive detection of phishing websites, especially newly created or dynamically generated phishing URLs.

**Real-Time URL Analysis**:

The system is designed to provide real-time analysis of URLs, ensuring immediate detection and feedback to users. This prevents delays in identifying phishing websites, reducing the likelihood of successful phishing attacks.

**Automated Response System**:

The proposed system will also provide an automated response mechanism to alert users or block access to malicious websites. This real-time feedback minimizes user exposure to potential threats.

**Continuous Learning**:

The system incorporates continuous learning, allowing it to adapt to new phishing techniques. The models are periodically retrained using newly gathered data to maintain accuracy as phishing tactics evolve. The requirement analysis for this work is divided into several key categories: **functional** requirements, **non-functional** requirements, **data** requirements, and **technical** requirements.

**Database Integration for Phishing Reports:**

The system allows users to manually report suspected phishing URLs, which are stored in a central database. This data is periodically exported as a CSV file, preprocessed, and used to retrain the machine learning model. The updated model is then integrated into the system, ensuring continuous improvement in phishing detection based on user feedback.

## 3.3.Functional Requirements

The functional requirements define the specific tasks that the phishing detection system must perform. These tasks directly contribute to the system's core functionality, ensuring it meets its intended purpose.

**URL Scanning and Feature Extraction:**

The system must be able to scan URLs and extract relevant features such as the length of the URL, domain structure, subdomain usage, the presence of numeric characters, special characters, and whether the URL uses HTTPS.

**Machine Learning Integration:**

The system must utilize machine learning models (e.g., Decision Tree, Random Forest, Extra Trees) to classify URLs as either phishing or legitimate based on the extracted features.

**Real-Time Detection:**

The system must process and classify URLs in real-time, providing immediate feedback to users on the legitimacy of the URL. Upon detecting a phishing URL, the system should automatically provide alerts to the user, potentially preventing access to malicious websites.

**Continuous Learning:**

The system should be designed to adapt to new phishing patterns through retraining or online learning, allowing it to stay effective as new phishing techniques evolve.

**Authenticated Users:**

Authenticated users are allowed to report phishing websites manually, this reported website must uploaded in the database and ML model should analyze and decide the result.

## 3.4.Non-Functional Requirements

Non-functional requirements describe the attributes the system must satisfy beyond its basic functionality. These attributes ensure the system operates effectively under real-world conditions.

**Performance:**

The system must be capable of analyzing and classifying URLs with minimal latency to ensure real-time detection. The response time should not exceed a few milliseconds per URL to prevent delays in user activity.

**Scalability:**

The system must be scalable to handle an increasing number of URLs as well as a larger volume of users. This ensures that the detection service can expand without degrading performance.

**Security:**

The system must ensure that the user’s interaction with the system, such as URL submissions and responses, is handled securely. Data exchanges must be encrypted, especially when dealing with sensitive information.

**Accuracy:**

The machine learning models must be trained to maintain a high level of accuracy, minimizing both false positives (legitimate URLs flagged as phishing) and false negatives (phishing URLs not detected).

**Usability:**

The system must be user-friendly, allowing users with minimal technical knowledge to easily interact with the URL checking tool. Feedback and alerts should be clear and concise, enabling users to understand the results of the phishing detection process.

**Availability:**

The system must be available for use 24/7 to ensure that users can continuously rely on it for protection against phishing attacks.

## 3.5.Data Requirements

The effectiveness of the phishing detection system relies heavily on the quality and variety of the data it processes. The system must have access to various data sources to accurately detect phishing URLs.

**URL Data:**

The system must be capable of analyzing various types of URLs, including those from legitimate websites and phishing websites. The data should include diverse patterns of phishing URLs to ensure that the model can generalize across different forms of phishing attacks.

**Feature Set:**

The key features extracted from the URLs should include:

- URL length
- Domain name and subdomain structure
- The presence of numeric characters and special characters
- The use of HTTPS
- Query parameters and path complexity

**Training Data:**

A substantial and balanced dataset of both phishing and legitimate URLs must be used to train the machine learning models. The dataset should be periodically updated to include new phishing patterns and website structures.

## 3.6.Technical Requirements

The technical requirements outline the necessary tools, frameworks, and infrastructure needed to develop, deploy, and maintain the phishing detection system.

**System Requirements:**

**CPU:**

**Minimum:** Multi-Core Processor(Intel i5 or AMD Ryzen5).

**Recommended:** High-End CPU (Inter i7/i9 or AMD Ryzen 7/9).

**RAM:**

**Minimum:** 8GB.

**Recommended:** 16GB or more for large datasets.

**GPU:**

**Minimum:** NVIDIA GPU with minimum of 4GB VRAM (GTX 1050, GTX 1650).

**Recommended:** NVIDIA GPU with 8GB VRAM for faster training (RTX 2060, RTX 3060).

**Storage:**

**Minimum:** 256GB, **Maximum:** 512GB.

**Software Requirements**

**Operating System**

**Development:** Windows 10/11, Ubuntu 20.04+, or any Linux-based OS

**Deployment:** Ubuntu Server (preferred for cloud hosting)

**Programming Languages**

**Python 3.8+ –** for backend and machine learning model development

**JavaScript (ES6+)** – for client-side scripting

**HTML5 & CSS3 (TailwindCSS)** – for user interface design

**Frameworks and Libraries**

**FastAPI** – for building high-performance REST APIs

**Scikit-learn** – for implementing ML models (Decision Tree, Random Forest, Extra Trees)

**Uvicorn –** ASGI server for running FastAPI in production

**SQLAlchemy** – ORM for database operations

**OAuthlib / Authlib –** for Google OAuth2.0 authentication

**Pydantic –** for data validation and schema definitions

**Database**

**SQLite –** for development and testing (lightweight and file-based)

**PostgreSQL** – for production-level deployments (scalable and robust)

**Development Tools**

**Visual Studio Code / PyCharm –** for code development

**Git –** version control

**Postman** – for API testing

**Browser Extension Tools**

**Chrome / Firefox Developer Tools –** for testing and debugging the extension

**Manifest V3 / WebExtension APIs** – for building browser extensions

**Cloud Hosting Platform**

**Render.com / Railway.app / Heroku –** for deploying FastAPI server and hosting the database

**Machine Learning Frameworks:**

The system will use machine learning libraries such as **scikit-learn** for model training and prediction. Additionally, **joblib** will be used to serialize and load the trained models efficiently in a production environment.

**Web Framework:**

The system will be deployed as a web extension that interacts with a backend powered by FastAPI, a modern and high-performance web framework for building APIs. FastAPI endpoints will handle user requests, including URL submissions, process them through the trained phishing detection model, and return results in real-time.

**API Development**:

A RESTful API will be developed to allow third-party applications or external systems to access the phishing detection service. This API will take URLs as input and return phishing detection results in real-time.

**Database Management:**

The system will require a database to store URL records, user interactions, and detected phishing URLs. A lightweight and scalable database **PostgreSQL** will be used.

**Security Measures:**

**OAuth Authentication:** The system integrates Google OAuth for secure user authentication, ensuring that only verified users can access certain features.

**HTTPS Encryption:** All communications between the client and server are encrypted to prevent data interception.

**API Security:** Secure API keys are used to restrict unauthorized access.

## 3.7.User Requirements

The system must be designed with the user's expectations and ease of use in mind. It should be intuitive and accessible to non-technical users who need real-time protection from phishing attacks.

**Instant Feedback:**

Users should receive immediate results when submitting a URL, indicating whether the URL is safe or potentially malicious.

**Clear Notifications:**

If a phishing URL is detected, the system should provide clear notifications to the user, detailing the risks and recommending actions (e.g., avoiding the link or blocking access).

**Mobile Compatibility:**

The system must be accessible from both desktop and mobile devices to provide flexibility for users on different platforms. This requirement analysis serves as the foundation for the design and development of the phishing detection system. By clearly defining the functional, non-functional, data, and technical requirements, the system will meet its objectives effectively, providing users with a powerful tool to combat phishing threats in real-time.

Users must consider using browsers like **Kiwi** Browser, **Yandex** Browser, **Firefox**, or **Samsung Internet**, which **support extensions**, including **Chrome extensions**.

# CHAPTER-4

# SYSTEM DESIGN

The system design outlines the architecture and components of the real-time phishing detection system. The system consists of several key modules, each responsible for specific tasks, including data processing, feature extraction, machine learning classification, and real-time URL analysis. By designing a modular architecture, the system is structured to perform real-time phishing detection with high accuracy and scalability.

## 4.1.Overview of the System

The System operates as a browser-integrated tool via a web extension, allowing users to scan URLs in real-time as they browse the internet. The system consists of multiple interconnected components including a FastAPI-powered backend server, an ensemble-based machine learning model, a secure database, and a user-friendly dashboard interface.

Upon scanning a URL, the system extracts relevant features, passes them to the trained model for prediction, and promptly returns a verdict—either phishing or legitimate. Authenticated users can view their scan history, report suspicious links, and access detailed results through the dashboard.

Additionally, administrators are provided with control features to manage user access, analyze reported URLs, and periodically update the model based on new phishing data to ensure continuous improvement in detection accuracy.

## 4.2.System Architecture

### User Interface (UI)

The front-end interface that allows users to submit URLs for phishing detection. Provides users with a simple and intuitive way to interact with the system, displaying results in real time.

**URL Submission API**

A RESTful API built with FastAPI that handles URL submissions from users. Receives URLs from the UI, forwards them to the feature extraction module, and returns detection results.

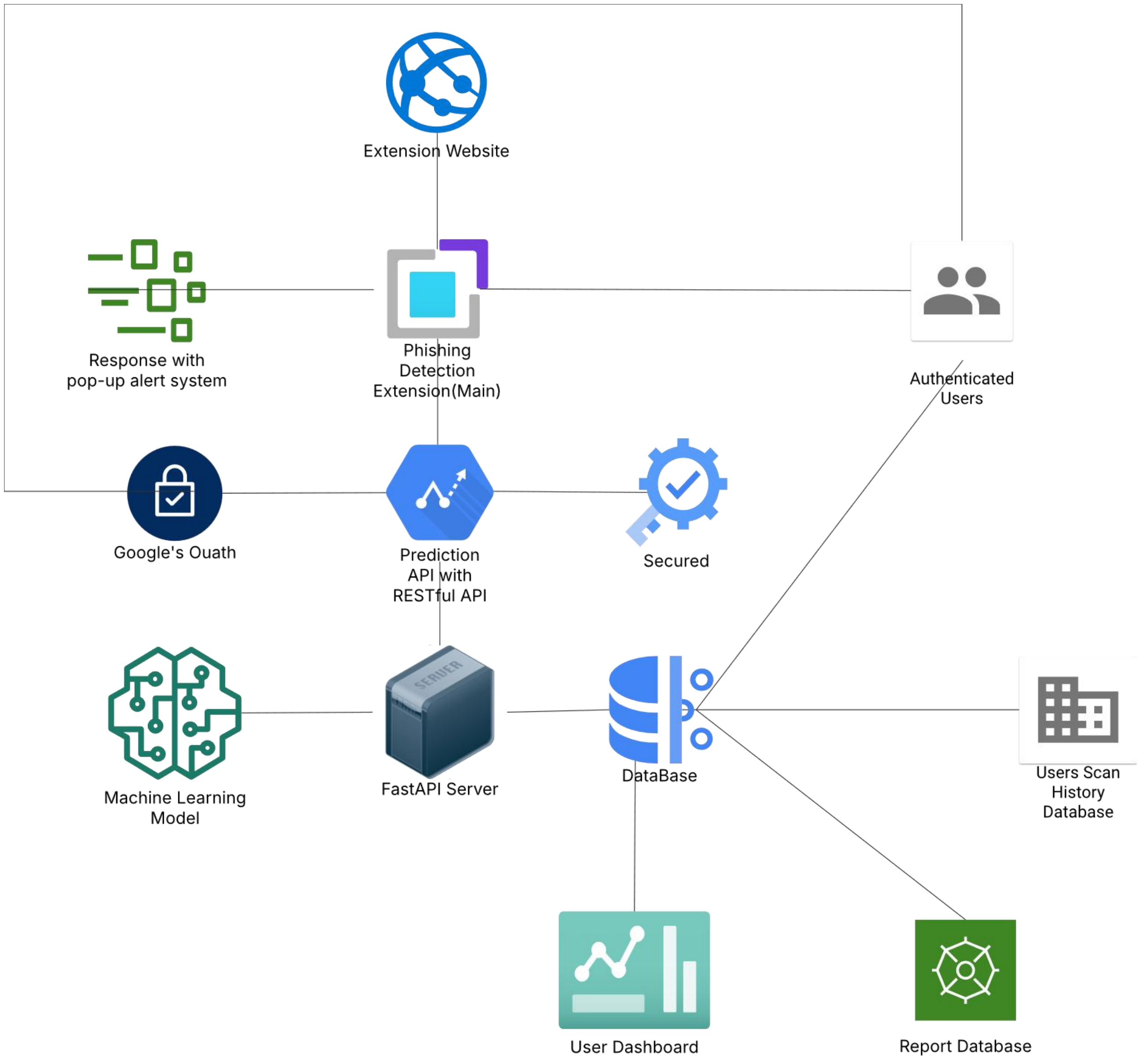


FIGURE 4.1 – ARCHITECTURE DIAGRAM

**Features Extracted**:

URL length, domain structure, subdomain count, numeric characters, special characters, HTTPS usage, and query parameters. Converts raw URL data into structured features for input to the machine learning models.

**Machine Learning Models**

A set of pre-trained machine learning models, including Decision Tree, Random Forest, and Extra Trees classifiers. Each model analyzes the extracted features and predicts whether the URL is phishing or legitimate.

A pipeline for updating and retraining the machine learning models with newly gathered data. Enables the system to adapt to new phishing techniques by periodically retraining models with recent data.

**Real-Time Detection Engine**

The core processing unit that coordinates feature extraction, model prediction, and result generation in real time. Ensures fast and efficient processing, generating immediate feedback to the user on URL legitimacy.

**Database**

PostgreSQL for storing URL records, detection results, and user reported URLs. Keeps records of analyzed URLs, which are useful for retraining models and improving detection accuracy over time.

**Authentication Server (Google Oauth)**

The system uses OAuth 2.0 with Google to authenticate users securely. When a user logs in, they are redirected to Google’s login page. After successful authentication, the system receives an access token and retrieves user details like name, email, and profile picture.

If the user is new, their data is stored in the database. Existing users are recognized, and their sessions are updated. All database operations are managed using SQLAlchemy sessions.

The authentication module is integrated into the FastAPI backend and protects API endpoints using token-based access. It ensures secure login, session management, and role-based access control.

**Result and Alert System**

Provides real-time feedback and alerts to users on the status of submitted URLs via web extension. Sends a clear phishing alert if the URL is flagged as malicious or a confirmation if it’s legitimate, enhancing user security.

## 4.3.Module Description

### 4.3.1.User Authentication Module

The User Authentication Module ensures that only verified users can access and interact with the system. It uses Google OAuth 2.0 to streamline the login process and enhance security. Upon login, the user is authenticated through Google's platform, and details such as name, email, and profile picture are fetched.

New users are added to the database, while returning users are identified and granted access. The system uses token-based session management, and all authentication logic is handled via FastAPI endpoints with support from OAuth libraries and SQLAlchemy for secure database transactions.

This module is critical for safeguarding user-specific features like URL scanning, viewing reports, and accessing dashboards.

### 4.3.2.URL Submission and Scanning Module

The URL Scanning Module is the core component responsible for analyzing submitted URLs and determining whether they are legitimate or phishing attempts. When a user inputs a URL through the browser extension or web interface, the system extracts relevant features such as URL length, domain structure, presence of HTTPS, special characters, and more.

These features are then passed to a pre-trained machine learning ensemble model consisting of classifiers like Decision Tree, Random Forest, and Extra Trees. The model processes the input and returns a prediction in real-time.

This module operates through FastAPI endpoints, ensuring fast and secure communication between the frontend and the backend model. All scan results can be logged and stored in the database for future reference or retraining.

### 4.3.3.Feature Extraction Module

The Feature Extraction Module is a critical backend component that processes raw URL input and derives meaningful attributes necessary for phishing detection. When a user submits a URL, this module parses it and extracts a structured set of features that the machine learning model can understand.

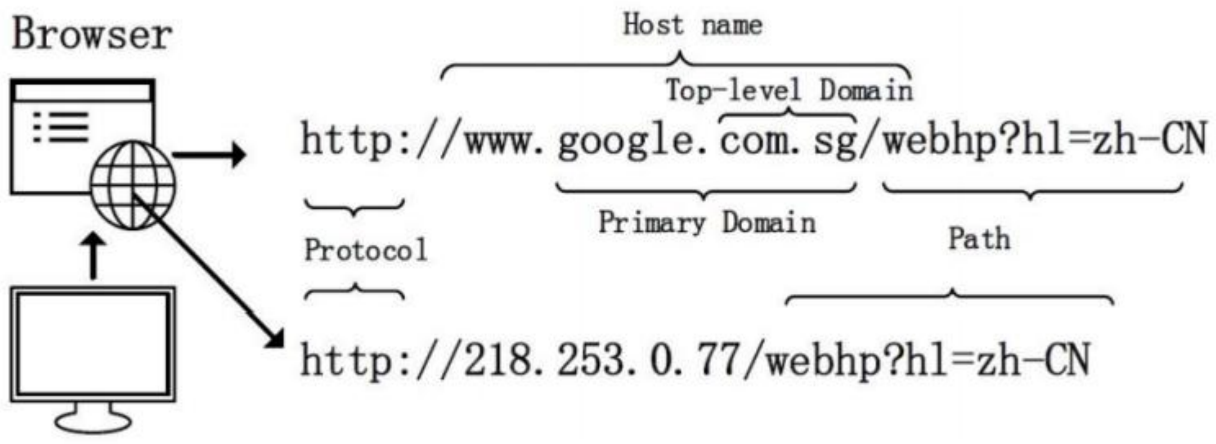

Browser
Host name
Top-level Domain
http://www.google.com.sg/webhp?hl=zh-CN
Protocol
Primary Domain
Path
http://218.253.0.77/webhp?hl=zh-CN

FIGURE 4.2 – URL FEATURES

The extracted features include:

- URL length
- Use of IP address instead of domain name
- Presence of special characters (e.g., @, -, =)
- Use of HTTPS protocol
- Number of subdomains
- Length and structure of the path and query string

This module ensures consistent formatting and sanitization of URLs before they are passed into the prediction pipeline. It improves model performance by standardizing the input and reducing noise. The module is developed as a reusable Python function or class, enabling integration into both real-time scanning and offline training workflows.

### 4.3.4.Machine Learning Prediction Module

The Ensemble Prediction Module is responsible for determining whether a submitted URL is legitimate or phishing by leveraging the combined strength of multiple machine learning models. Instead of relying on a single model, this module uses an ensemble approach that aggregates the predictions of several classifiers such as Decision Tree, Random Forest, and Extra Trees Classifier.

Each model analyzes the extracted features independently and outputs a prediction. These predictions are then passed to a voting mechanism—either majority voting or weighted voting—to produce the final decision. This ensemble strategy enhances accuracy and robustness by reducing individual model biases and improving generalization on unseen data.

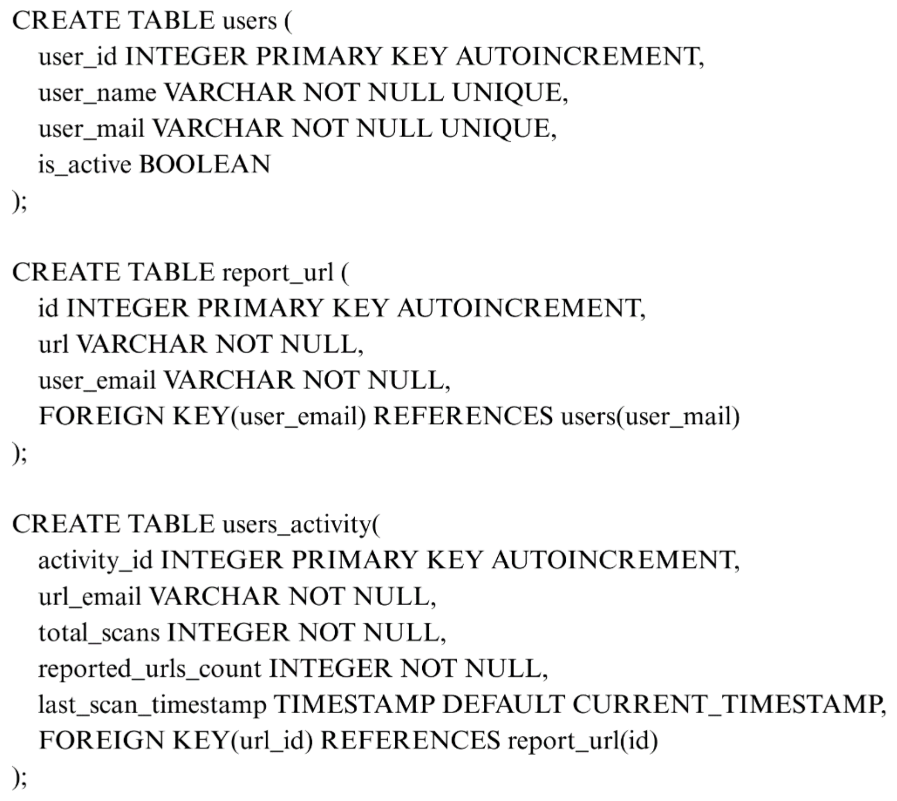
The ensemble module is optimized for real-time performance, allowing it to deliver predictions quickly when integrated with the FastAPI backend or browser extension interface.

### 4.3.5 Database Module

The Database Module manages all data storage operations in the system. It is responsible for securely storing information such as authenticated user details, scanned URLs, prediction results, reported phishing URLs, and activity logs. This module interacts with the backend services using ORM (Object Relational Mapping) tools like SQLAlchemy to perform efficient database transactions.

For user authentication, it stores OAuth-based user profiles, including email, username, and tokens. When a user submits a URL for scanning, the system logs the URL, prediction result, and timestamp into the database. Additionally, if a URL is reported by the user, the module saves the report with metadata to allow model retraining or administrative review.

**SCHEMA**



```
CREATE TABLE users (
   user_id INTEGER PRIMARY KEY AUTOINCREMENT,
   user_name VARCHAR NOT NULL UNIQUE,
   user_mail VARCHAR NOT NULL UNIQUE,
   is_active BOOLEAN
);

CREATE TABLE report_url (
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   url VARCHAR NOT NULL,
   user_email VARCHAR NOT NULL,
   FOREIGN KEY(user_email) REFERENCES users(user_mail)
);

CREATE TABLE users_activity(
   activity_id INTEGER PRIMARY KEY AUTOINCREMENT,
   url_email VARCHAR NOT NULL,
   total_scans INTEGER NOT NULL,
   reported_urls_count INTEGER NOT NULL,
   last_scan_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY(url_id) REFERENCES report_url(id)
);
```

FIGURE 1 DATABASE SCHEMA

Role-based access is also handled through this module, where regular users have limited access to their data and reports, while administrators have extended privileges to manage user data, export datasets, and oversee reports. The module supports PostgreSQL in production for its scalability, though SQLite can be used during development.

### 4.3.6.API Integration Module

The API Integration Module serves as the core communication layer between the frontend web extension, external systems, and the backend services of the phishing detection system. It is developed using FastAPI, a modern, high-performance web framework tailored for building RESTful APIs with Python.

This module defines endpoints that handle various operations such as submitting URLs for scanning, fetching user-specific data, accessing report logs, and managing administrative tasks. Each API endpoint is designed to accept input (e.g., URLs) via HTTP requests and return a structured JSON response containing the phishing detection results or relevant data.

Security is enforced through token-based authentication mechanisms and Google OAuth integration. The module ensures that only authenticated users can access protected endpoints and that sensitive operations are restricted to users with the appropriate roles.

The APIs are also designed to support asynchronous execution, improving response times and scalability. With clearly defined routes and responses, the module allows smooth integration with third-party tools or browser extensions for real-time phishing detection.

### 4.3.7.Reporting and Admin Module

The Reporting and Admin Module is responsible for managing user reports, monitoring system usage, and administering key functionalities of the phishing detection platform. Authenticated users can report suspicious URLs, which are logged into the database for further analysis. These reported entries contribute to refining the detection model and expanding the phishing URL dataset.

Users have access to a dashboard where they can view their scan history, report statuses, and contributions to the system. The dashboard provides transparency and encourages user participation in identifying malicious content on the web.

From an administrative perspective, this module enables privileged users (admins) to manage user accounts, review reported URLs, and take corrective actions if necessary. Admins can also export datasets, update the machine learning model using newly reported data, and monitor system performance.

The module is secured through role-based access control (RBAC), ensuring that only authorized users can access administrative functionalities. It plays a critical role in keeping the system dynamic, up-to-date, and aligned with evolving phishing techniques.

### 4.3.8.Web Extension Module

The Web Extension Module serves as a user-facing component that seamlessly integrates phishing detection capabilities into the user's browsing experience. This browser extension allows users to scan any active webpage or a manually entered URL in real-time without leaving their browser environment.

When a user interacts with the extension, it captures the current tab's URL or accepts user input, then sends this data to the backend via the FastAPI endpoint. The backend processes the URL, extracts features, performs phishing prediction using the trained ensemble model, and returns the result to the extension interface.

The extension displays a clear message (e.g., "Safe", "Suspicious", or "Phishing") based on the model's prediction, helping users make informed decisions before interacting with the content of the page.

It is lightweight, built using modern web technologies, and designed to be responsive and user-friendly. Additionally, it provides access to scanning history for authenticated users, aligning well with the system's focus on usability, transparency, and security.

## 4.4.SYSTEM DESIGN DIAGRAMS

### 4.4.1.Context Diagram

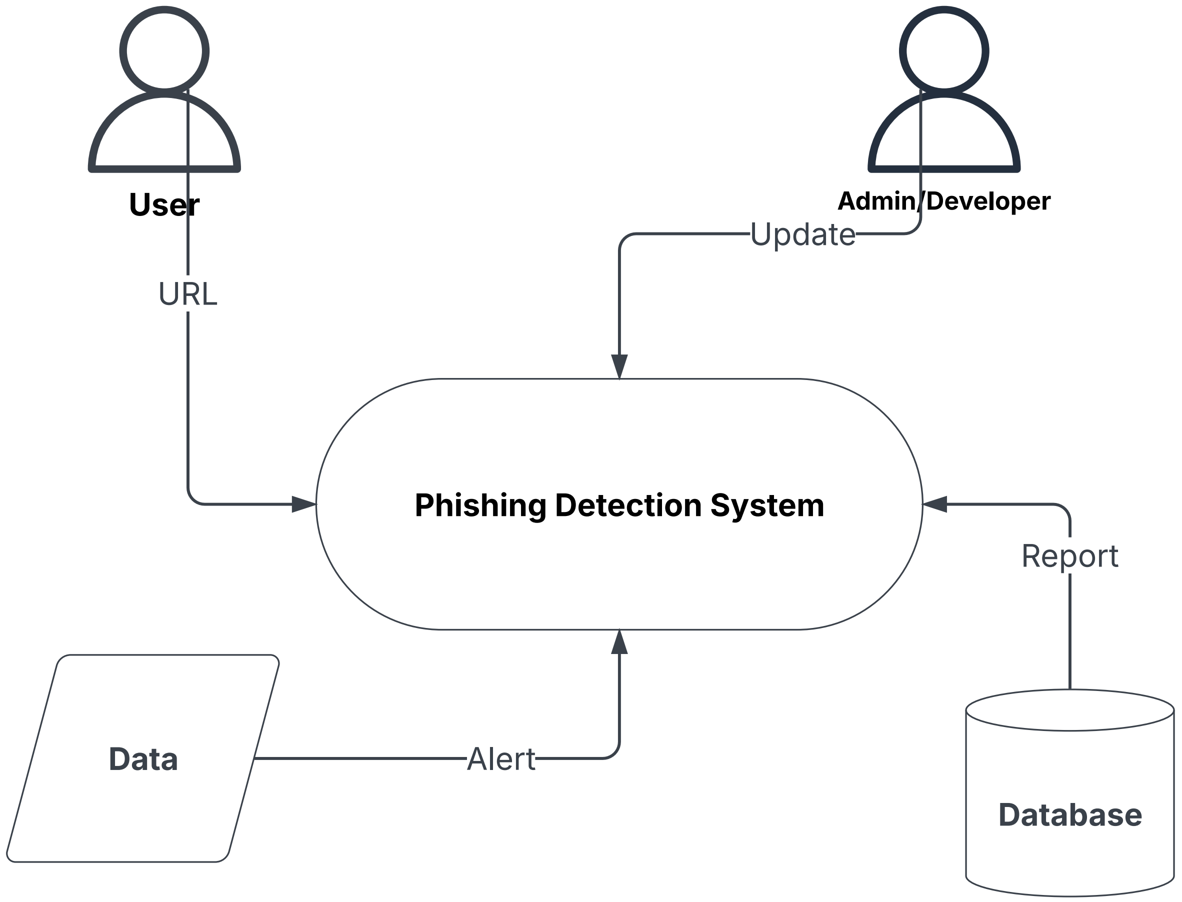


FIGURE 4.3 – CONTEXT DIAGRAM

The context diagram provides a high-level overview of how different entities interact with the Phishing Detection System. It illustrates the flow of information between users, data sources, and the system.

**Entities & Interactions:**

**User (Left-Side - Inputs a URL)**

The user enters a URL to check if it is phishing or safe.

The system processes the URL, extracts relevant features, and predicts its legitimacy.

**Phishing Detection System (Core Component)**

This is the main processing unit where the system analyzes incoming URLs.

It performs feature extraction and prediction using machine learning models.

**Data (Bottom-Left - Feeds into the System)**

The system relies on external datasets, previously reported URLs, and machine learning models to enhance detection.

This data is continuously used for improving phishing identification.

**Alert (System Output - Bottom)**

If a URL is flagged as phishing or suspicious, an alert is generated.

This can be displayed to the user via a web extension or website notification.

**Database (Bottom-Right - Stores Reports & Updates)**

The system stores reports on detected phishing URLs in the database.

This helps in maintaining a record of flagged URLs for future reference.

**User (Right-Side - Updates & Reports URLs)**

Users can report phishing URLs to contribute to the database.

Admins or authorized users can update the system, improving detection accuracy.

This diagram emphasizes the system's interaction with users, data sources, and the database, ensuring real-time phishing detection and continuous learning.

### 4.4.2.Use Case Diagram

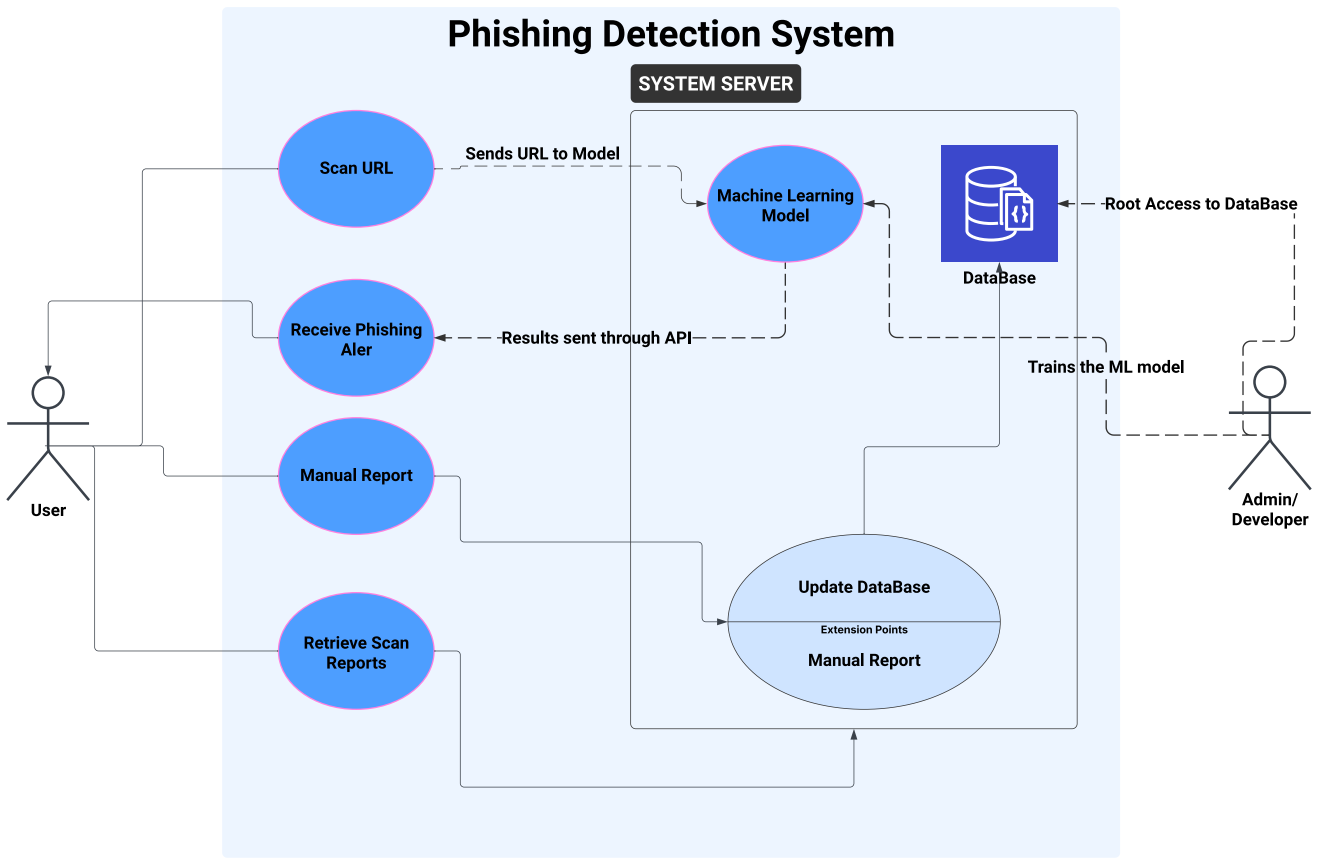


FIGURE 4.4 – USE CASE DIAGRAM

Use Case Diagram Explanation - Phishing Detection System

This Use Case Diagram represents the interactions between users and the system, outlining key functionalities.

**Actors:**

**User (Left & Right)**

A regular user who interacts with the system to scan URLs, receive phishing alerts, manually report phishing URLs, and retrieve scan reports.

**System Server (Central Component)**

Handles user requests, processes URLs, and interacts with the machine learning model and database.

**Use Cases & Interactions:**

**Scan URL:**

The user submits a URL to check if it is phishing or safe.

The system sends this URL to the Machine Learning Model for classification.

**Receive Phishing Alert:**

Once the model processes the URL, the system provides feedback.

The user gets an alert if the URL is detected as phishing.

**Manual Report:**

Users can report a URL if they suspect it to be phishing.

This report is added to the database for future analysis.

**Retrieve Scan Reports:**

Users can view previously scanned URLs and their classification results.

**Machine Learning Model Interaction:**

The ML model processes submitted URLs and classifies them.

The Database stores new phishing reports and helps train the model for improved accuracy.

**Update Database:**

The database is updated with newly reported URLs.

The system administrator or backend service updates the model periodically to enhance detection performance.

This diagram illustrates a well-structured phishing detection workflow, ensuring real-time analysis, reporting, and continuous model improvement.

### 4.4.3.Data Flow in the System

#### User Interaction (Web Extension)

The user interacts with the phishing detection system through a web extension. The extension captures the URL the user is visiting and sends it to the backend server for analysis.

#### Backend Processing

The FastAPI server receives the URL request from the web extension. The backend then preprocesses the URL and extracts relevant features such as domain structure, length, special characters, and security indicators.

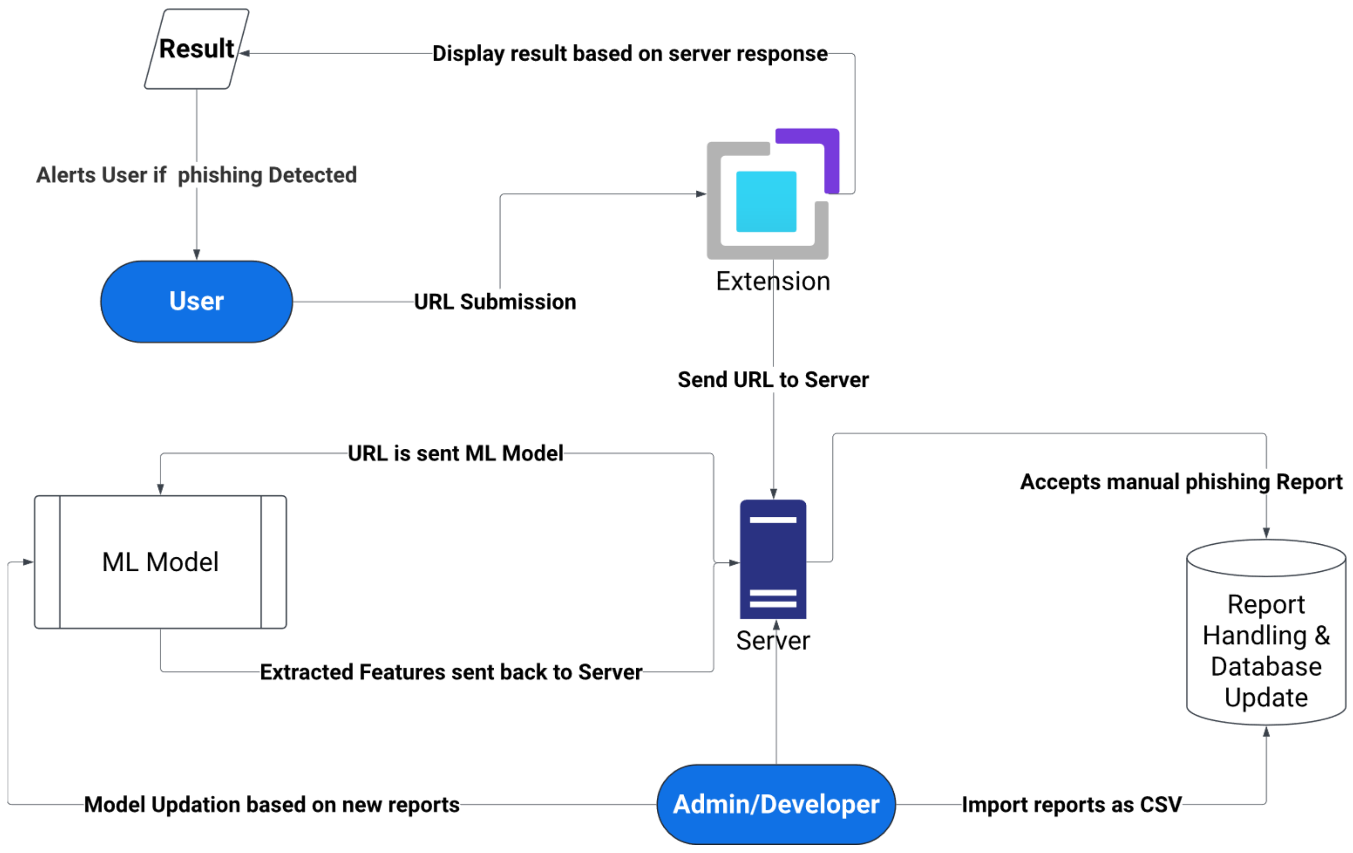

Result
Display result based on server response
Alerts User if phishing Detected
User
URL Submission
Extension
Send URL to Server
URL is sent ML Model
ML Model
Accepts manual phishing Report
Server
Report Handling & Database Update
Extracted Features sent back to Server
Model Updation based on new reports
Admin/Developer
Import reports as CSV

FIGURE 4.5 – DATA FLOW IN THE SYSTEM (HIGH LEVEL DFD)

**Machine Learning Model Prediction**

The preprocessed URL data is sent to the ML Model, which predicts whether the URL is phishing or legitimate. The model returns a classification result (phishing or safe) to the backend server.

**Server:**

A uvicorn server runs on a port:8000 for the backend. It also uses cors middleware to avoids arising **CORS** issues from the front-end port.

**Displaying Results to the User**

The backend server sends the prediction result back to the web extension. If the URL is phishing, the system triggers a warning alert to notify the user. Otherwise, the user can proceed safely.

**Admin/Developer:**

Admin and Developer can export the report as CSV file and uses that data to train the Machine Learning Model.

**Report Handling & Database Update**

The phishing detection system maintains a database to store analyzed URLs, user interactions, and reports. If a URL is confirmed as phishing, the system updates the database with this new information. The Admin/Developer can access this data to monitor threats, improve detection accuracy, and update the model periodically.

### 4.4.4. Database Design (ER Diagram)

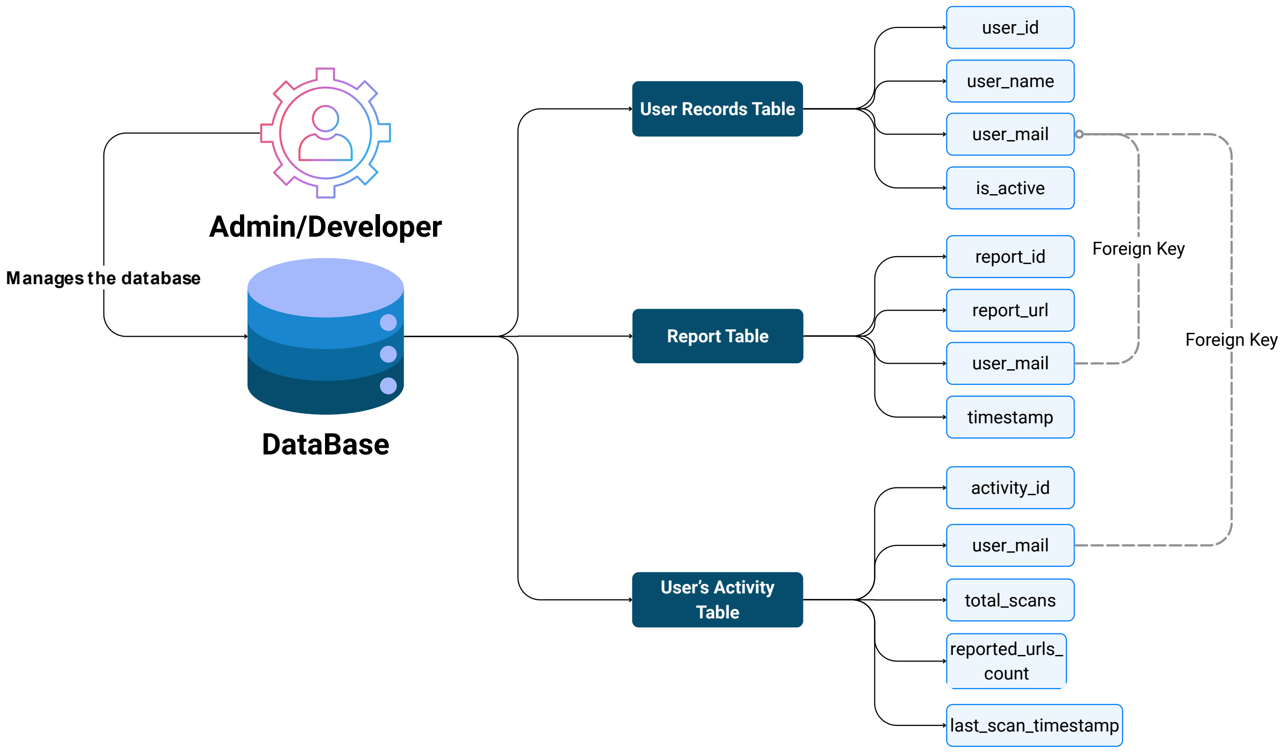


FIGURE 4.6 – ENTITY RELATIONSHIP DIAGRAM

**User Records Table:**

This table stores registered users' details and their account status.

| Field | Description |
|---|---|
| user_id | A unique identifier for each user (Primary Key). |
| user_name | The full name of the user. |
| user_mail | The user's email (used for login and tracking). Should be unique. |
| is_active | Boolean value indicating if the user account is active. |

**Report Table:**

This logs every URL scan or report submitted by a user.

| Field | Description |
|---|---|
| report_id | Unique ID for each report (Primary Key). |
| report_url | The URL reported/scanned by the user. |
| user_mail | Email of the user who submitted the report (acts as a foreign key, links to User Records Table). |
| timestamp | Date and time when the report was submitted. Useful for chronological tracking. |

**User's Activity Table:**

This tracks **statistics and activity metrics** for each user, making it useful for dashboards or analytics.

| Field | Description |
|---|---|
| **activity_id** | Unique ID for activity record (Primary Key). |
| **user_mail** | Email of the user (acts as a foreign key). |
| **total_scans** | Total number of URLs scanned by the user. |
| **reported_urls_count** | Number of those URLs that were flagged as suspicious or malicious. |
| **last_scan_timestamp** | Timestamp of the most recent scan. |

## 4.5.Technology Stack

### 4.5.1.Frontend: HTML**,** JavaScript and Tailwind CSS

HTML is the backbone of web content, used to structure the user interface. TailwindCSS, a utility-first CSS framework, was chosen to style the interface efficiently without writing custom CSS for every component. It promotes rapid UI development, consistency in design, and responsiveness across devices.

JavaScript is used for adding interactivity to the website. It enables features such as URL submission, dynamic content rendering, and handling asynchronous operations (like calling backend APIs). It’s lightweight and perfect for building fast and responsive web pages.

### 4.5.2.Backend:

FastAPI is a modern, fast (high-performance) web framework for building APIs with Python. It supports asynchronous programming, making it ideal for handling real-time scanning requests and delivering results efficiently.

### 4.5.3.Machine Learning:

The system uses Scikit-learn to train and serve machine learning models (Decision Tree, Random Forest, Extra Trees) for phishing detection. This library offers powerful tools for model development and evaluation.

### 4.5.4.Database:

**PostgreSQL:**

A relational database is used to store user data, scanned URLs, and reports. It supports ACID properties and ensures data integrity. Foreign key constraints and indexing improve performance and data consistency.

### 4.5.5.Authentication:

**Google OAuth:**

Google OAuth is implemented for secure and convenient user authentication. It allows users to sign in using their Google accounts without managing separate login credentials.

# CHAPTER 5

# IMPLEMENTATION

This chapter provides an in-depth overview of each module within the Real-Time Phishing Detection System, covering the coding frameworks, machine learning models, and methodologies applied throughout development.

## 5.1.Environment Setup

To ensure a smooth development experience and proper dependency management, the following environment setup was adopted for building the Real-Time Phishing Detection System:

**Integrated Development Environment (IDE):**

The project was developed using Visual Studio Code (VSCode), which provides powerful features such as syntax highlighting, linting, debugging support, and integrated terminal access, making it ideal for Python-based machine learning projects.

**Python Version:**

The system is implemented using Python 3.8+, which provides compatibility with modern libraries and features required for machine learning, API development, and data preprocessing.

**Virtual Environment Setup:**

To manage dependencies and avoid conflicts with global packages, a Python virtual environment (venv) is created and activated. This ensures that all required packages are installed and run in an isolated environment.

**Setup steps are:**

- Create virtual Environment.
- Activate the Virtual environment.
- Install the dependencies.

**Dependency Management:**

The requirements.txt file includes all the core dependencies used in the system, such as:

- pandas, numpy for data manipulation.
- scikit-learn for machine learning model development.
- fastapi, uvicorn for API development.
- joblib for model serialization.
- requests, python-multipart, and others as needed.

**OAuth Configuration: Google Client ID and Secret**

To enable secure user authentication, the system integrates Google OAuth 2.0, allowing users to sign in using their Google accounts. This provides a seamless and trusted authentication mechanism while offloading the burden of credential management to Google.

**Client ID and Client Secret:**

The Google OAuth Client ID and Client Secret are essential credentials obtained from the Google Developer Console. These values uniquely identify the application and are used to authenticate it with Google's servers.

**Security and Storage:**

For security purposes, these credentials are not hard-coded into the application. Instead, they are securely stored in a .env file (environment variable configuration file). This approach helps prevent accidental exposure or leakage of sensitive information, especially in public repositories.

**Functionality:**

**During the OAuth flow:**

The client ID and secret are used to initiate the OAuth process. Google verifies the credentials and authenticates the user. Upon successful authentication, the user is granted access to their Google profile information (such as email ID), which is used for login or registration within the system.

## 5.2.Dataset Preparation

**Dataset Source:**

The dataset for training and testing was sourced from public databases such as [source, e.g., Kaggle or PhishTank]. And Dataset is stored as a **CSV** format.

**Data Cleaning:**

Preprocessing steps included handling missing values, removing duplicates, and correcting any anomalies to improve data quality.

**Feature Engineering:**

Features extracted from URLs included: URL length, Domain and subdomain characteristics, Presence of special characters (e.g., "-", "@"), Numerical and alphabetical patterns

**Train-Test Split:**

The data was divided into training and testing sets with a typical 80/20 split, ensuring a balanced and representative dataset for model training and evaluation.

## 5.3.Model Development

**Models Used:** Decision Tree Classifier**,** Random Forest Classifier**,** Extra Trees Classifier.

**Model Selection:**

These models were selected based on factors such as interpretability, speed, and accuracy, making them suitable for real-time detection.

**Training Process:**

Each model was trained on the engineered URL features, and performance metrics (accuracy, precision, recall) were evaluated to select optimal parameters.

**Serialization:**

Trained models were saved as `.pkl` files using `joblib`, enabling seamless integration with the FastAPI for real-time inference.

## 5.4.Feature Extraction Module

**Objective:**

To extract relevant characteristics from each URL, providing structured input features for the machine learning models.

**Key Features Extracted:** URL length, domain length**,** Number of certain characters (e.g., "-", "@", "//")**,** Domain suffix and the number of digits.

**Implementation:**

The `extract_features_single(url)` function was developed to process URLs, extracting features and returning them as a vector for model input.

## 5.5.Ensemble Prediction Module

**Ensemble Decision Process:**

Combines predictions from the Decision Tree, Random Forest, and Extra Trees classifiers. Each model returns a probability score for the URL being phishing, and the average score is calculated. If the average probability exceeds a threshold (e.g., 0.4 for heightened sensitivity), the URL is classified as "Phishing."

## 5.6.API Development with FastAPI

**API Overview:**

A RESTful API was developed using FastAPI to allows seamless access to phishing detection model.

**Endpoints:**

**`/predict-url`** — Accepts a URL input and returns a classification result (Phishing or Benign) along with a confidence score.

**`/user`** — Here the admin/Developer can view and manage user details.

**`/report`** — This endpoint receives URL as input and then result of the URL is returned as output.

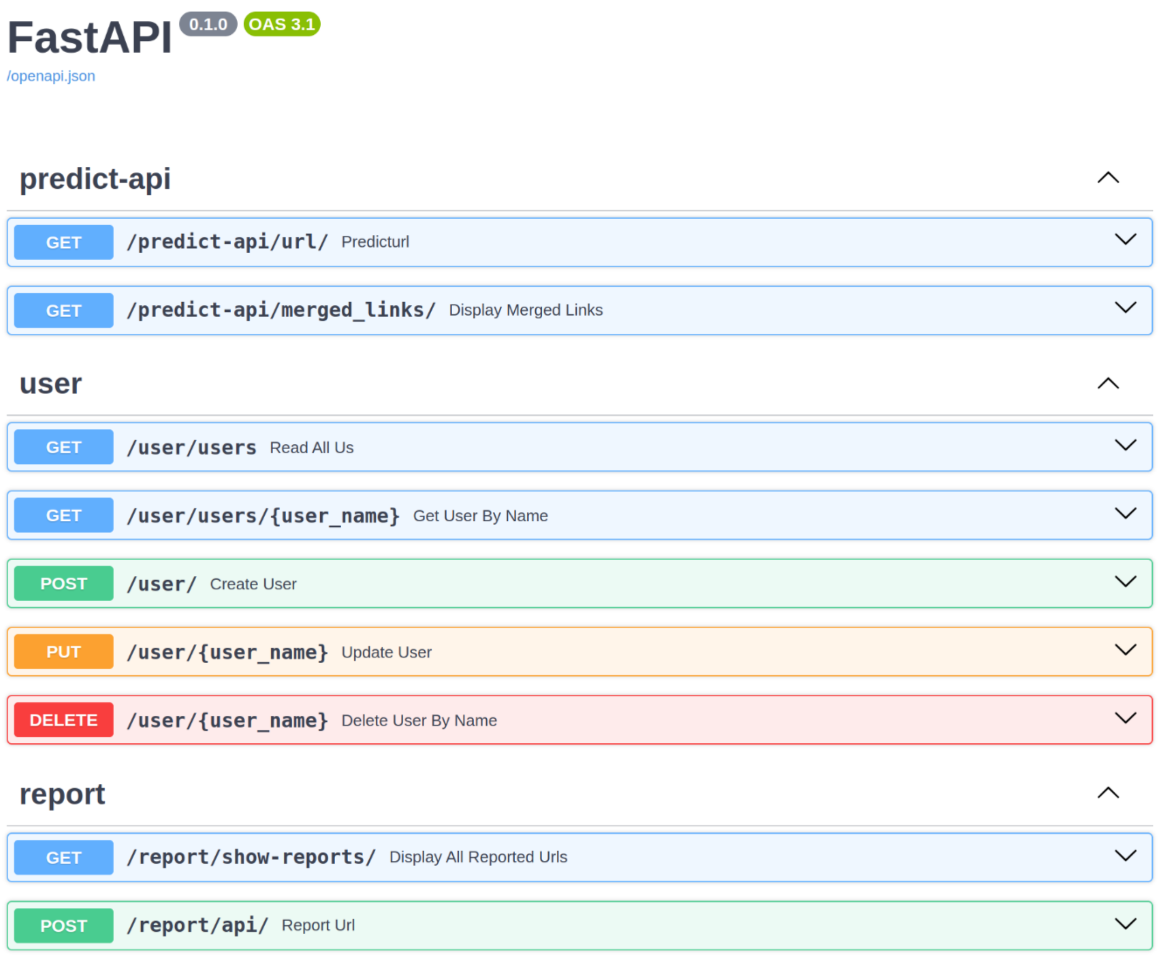
**`/scan-report`** — Currently our system doesn't have this endpoint, because we have planned to add in this future. This endpoint returns the user's dashboard data like (No.of URLs Scanned, No.of URLs reported by user, and more…).

**`/oauth`** — Google authentication endpoint, where user can login and logout and it also collects the details of authenticated user from google cloud using Oauth2.

**`/index `** — API's index page.

**Integration:**

Each request triggers the `ensemble_classify` function, which loads models, extracts features, and delivers a final classification. The model is exported as pickle (.pkl)format which is imported in the FastAPI server.


FastAPI 0.1.0 OAS 3.1
/openapi.json

predict-api

GET `/predict-api/url/` Predicturl

GET `/predict-api/merged_links/` Display Merged Links

user

GET `/user/users` Read All Us

GET `/user/users/{user_name}` Get User By Name

POST `/user/` Create User

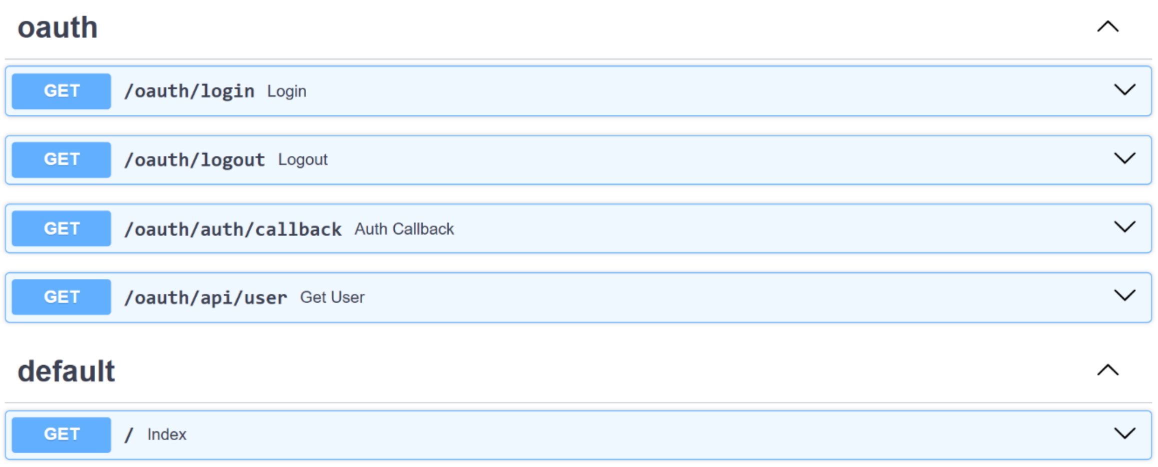
PUT `/user/{user_name}` Update User

DELETE `/user/{user_name}` Delete User By Name

report

GET `/report/show-reports/` Display All Reported Urls

POST `/report/api/` Report Url


oauth

GET /oauth/login Login

GET /oauth/logout Logout

GET /oauth/auth/callback Auth Callback

GET /oauth/api/user Get User

default

GET / Index

FIGURE 5.1 – API ENDPOINTS

## 5.7.Web Extension Integration

The web extension acts as the user interface, enabling seamless interaction with the phishing detection system. It allows users to scan URLs in real time directly from their browser, enhancing security and convenience. When a user clicks on the extension icon, they can either analyze the currently opened webpage or manually enter a URL for scanning. The extension then communicates with the FastAPI backend by sending the URL via an HTTP request and receiving the detection results.

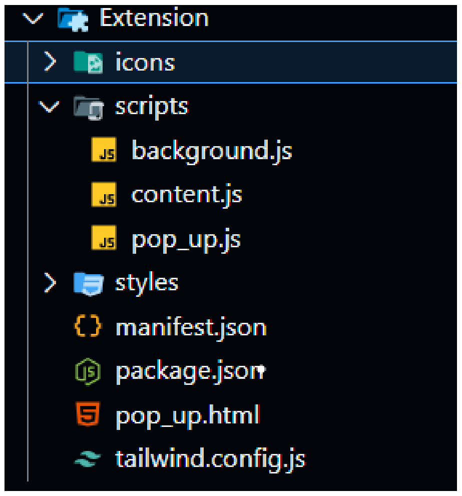

```
Extension
  icons
  scripts
    background.js
    content.js
    pop_up.js
  styles
  manifest.json
  package.json
  pop_up.html
  tailwind.config.js
```

FIGURE 5.2 EXTENSION FILE STRUCTURE

Based on the model's prediction, the extension provides real-time alerts, warning users if a URL is identified as phishing. This ensures immediate protection against potential threats. Additionally, the extension may store scanned URL data locally or transmit logs to the backend for further analysis and report generation. For features requiring authentication, such as submitting phishing reports, the system integrates Google OAuth to ensure secure user login and data access.

To further improve security, the extension can also implement automatic scanning, where it passively analyzes URLs in the background and alerts users about suspicious links before they interact with them. By integrating these functionalities, the web extension significantly enhances the usability of the phishing detection system, providing a proactive and real-time security mechanism within the browser. The **extension** is **published** in **Microsoft Addons, extension URL** is attached in **GitHub repo**.

## 5.8.Google OAuth Authentication

User authentication is implemented using OAuth 2.0 via Google, allowing secure login without passwords. Upon authentication, the system retrieves and stores user details such as name, email, profile picture, access token, and refresh token in the database. Database sessions in Python handle user data operations efficiently, ensuring seamless access and security. This approach enhances user experience while enabling role-based access control.

## 5.9.Database and Report Handling

Authenticated users can scan, report URLs and can track their activities in the dashboard. . They can also access the reported database if authorized. Administrators have additional privileges, including managing users, exporting data, and updating the phishing detection model using reported URLs to enhance accuracy.

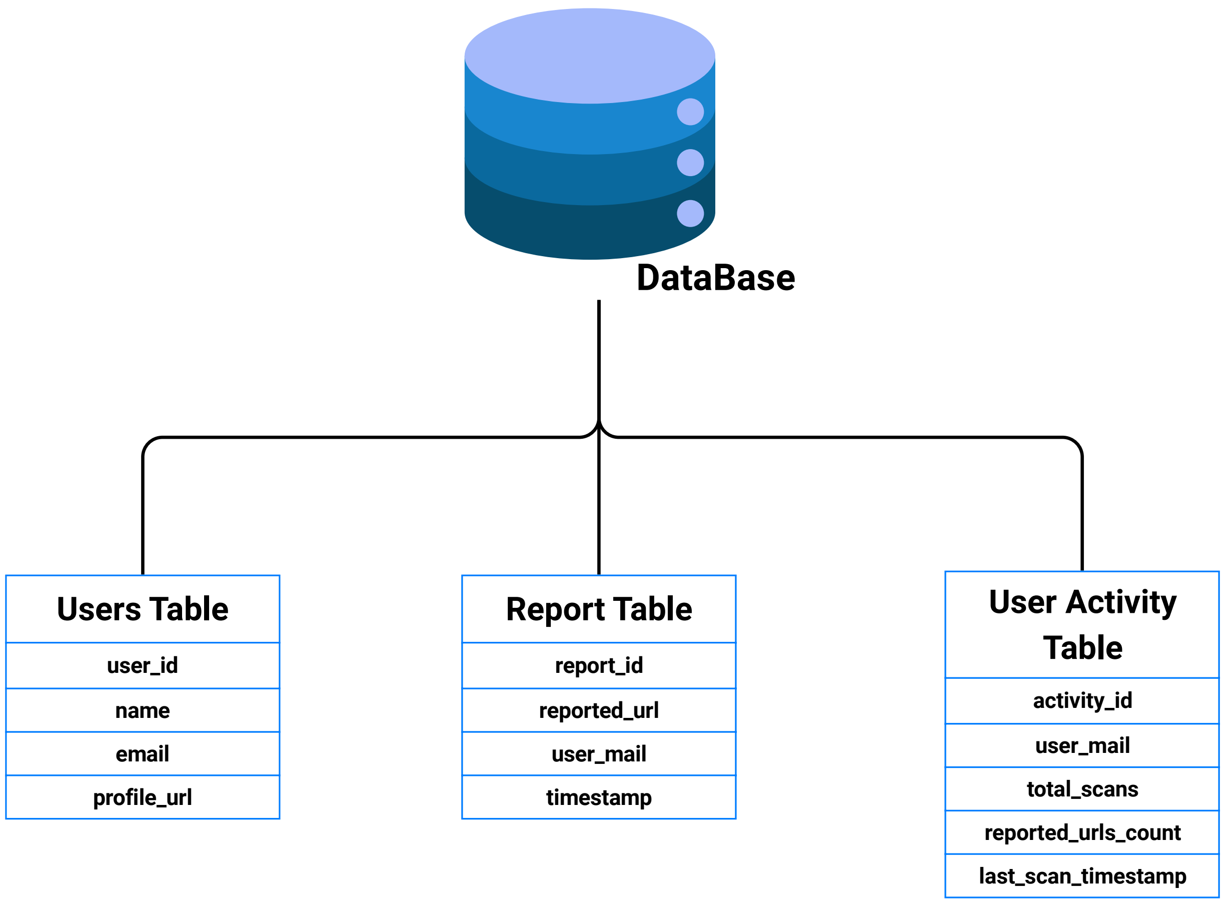


FIGURE 5.3 – STRUCTURE OF DATABASE

**Deployment and Hosting**

The FastAPI application is deployed and hosted on a cloud platform such as Render.com. The cloud environment provisions a database to store user data, scanned URLs, and phishing detection results. The deployed system follows the process outlined earlier, ensuring seamless execution of URL analysis and phishing detection.

## 5.10.Comparative Implementation Analysis

| Criteria | Existing Systems | Our System |
|---|---|---|
| **Detection Approach** | Mostly rely on static blacklists or heuristics | Uses real-time Machine Learning classification with ensemble models (Decision Tree, RF, Extra Trees) |
| **User Interface** | Limited or no interface, often backend only | Interactive frontend using HTML, TailwindCSS with real-time alerts via browser extension |
| **Backend Framework** | Typically use traditional frameworks like PHP or Django | Lightweight, high-performance backend using **FastAPI** |
| **AuthenticationMechanism** | Often absent or limited to basic login | Google OAuth2.0 authentication with session handling and DB integration |
| **Feature Extraction** | Basic features like domain name, URL length | Advanced features including subdomain depth, IP-based domain, special char counts, etc. |
| **User Contribution** | No support for user-reported URLs | Users can **report URLs**; admins can export and use them to **retrain the model** |
| **Deployment** | Hosted locally or not deployable at scale | **Deployed on cloud platform (e.g., Render)** with API endpoint for web extension |
| **Model Update Strategy** | Mostly static models | Periodic retraining based on reported data from authenticated users |
| **Integration** | Limited or no third-party integration | Can be integrated with web pages or extended as a browser plugin |
| **Extension Support** | Rarely provided | Chrome Extension built to **monitor active tab URLs** and show detection alerts |

# CHAPTER-6

# RESULTS & DISCUSSIONS

## 6.1.Introduction to Evaluation

The evaluation phase of the Real-Time Phishing Detection System focuses on validating the effectiveness and efficiency of the entire application, from machine learning model predictions to the user experience of interacting with the web extension. The primary goal of this evaluation is to ensure that the system reliably detects phishing URLs in real-time, secures user data through robust authentication, and maintains a seamless integration with cloud services and the database. Various components of the system—such as the model, API endpoints, database interactions, and frontend modules—are assessed individually and collectively. The evaluation also takes into consideration how well the system performs under different usage conditions, including varying types of URLs, different devices, and user load levels.

## 6.2.Model Performance Metrics

The effectiveness of the phishing detection model was evaluated using standard classification performance metrics: Accuracy, Precision, Recall, and F1-Score. These metrics help quantify the model’s ability to correctly identify phishing and legitimate URLs.

To evaluate the effectiveness of each model used in this system, we employed multiple performance metrics including accuracy, precision, recall, F1-score, and a confusion matrix for each classifier. These metrics provide a comprehensive understanding of how well the models distinguish between legitimate and phishing URLs across various categories.

### 6.2.1.Decision Tree Classifier Performance

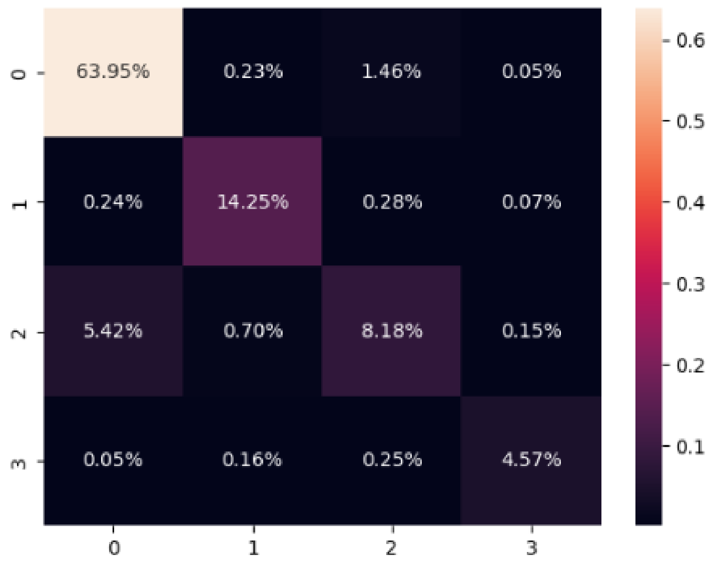
The Decision Tree Classifier yielded a test accuracy of 90.94%, which indicates a high level of correct classifications on unseen data.

```
##############################################
######-Model => <class 'sklearn.tree._classes.DecisionTreeClassifier'>
Test Accuracy :  90.94%
              Classification_report
              precision    recall  f1-score   support

           0       0.92      0.97      0.94     85565
           1       0.93      0.96      0.94     19319
           2       0.80      0.57      0.66     18805
           3       0.95      0.91      0.93      6550

    accuracy                           0.91    130239
   macro avg       0.90      0.85      0.87    130239
weighted avg       0.90      0.91      0.90    130239

             Confusion_matrix
```



```
###################- End -###################
```

FIGURE 6.1 – DECISION TREE CLASSIFIER REPORT

From the classification report, the model shows strong performance **for classes 0, 1, and 3 with high precision and recall values**. However, **class 2 has relatively low recall (0.57)**, which implies that many instances belonging to this class were misclassified. This is further visualized in the confusion matrix, where class 2 is often confused with class 0.

The confusion matrix heatmap confirms this trend:

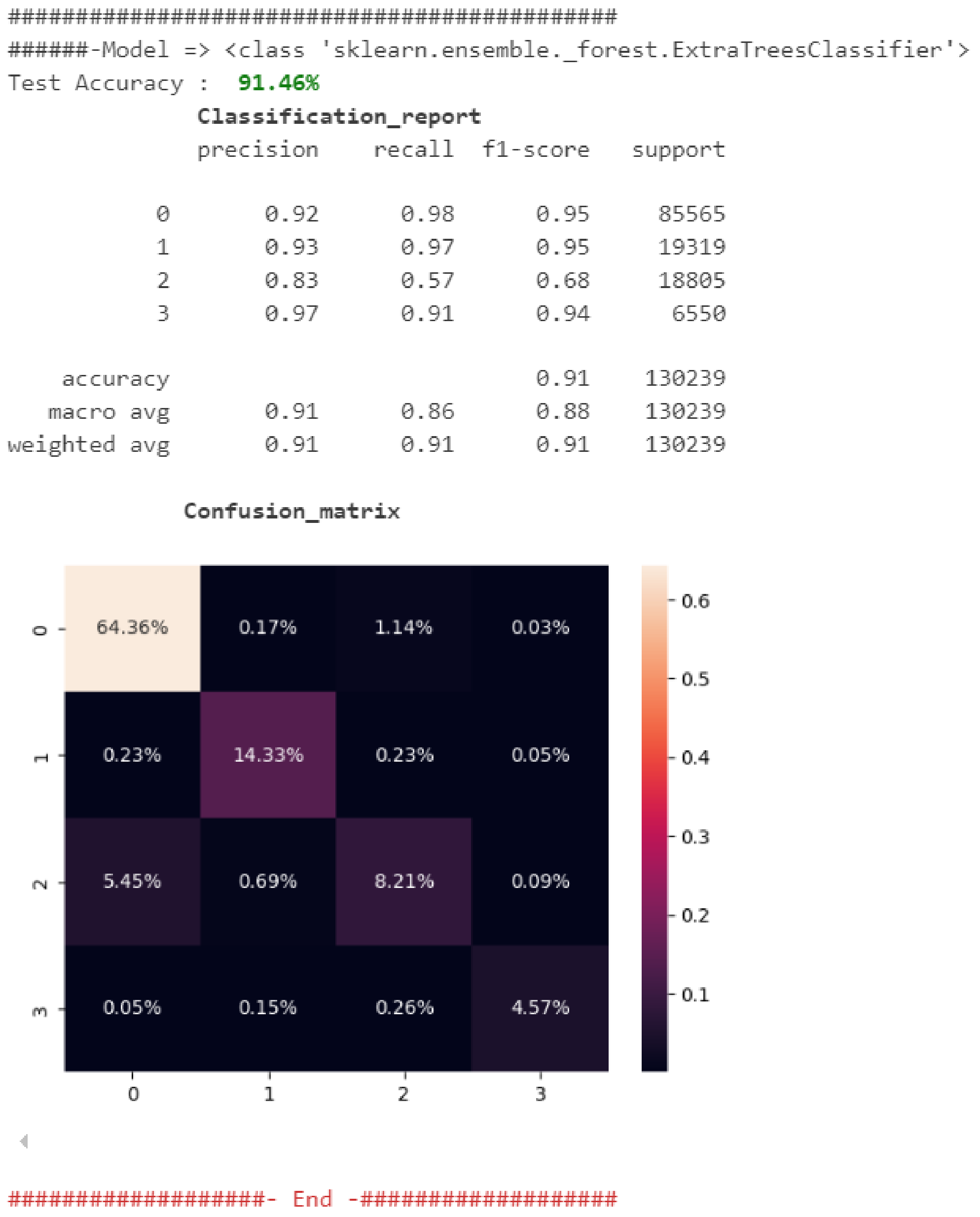
Class 0 (Legitimate): 63.95% correctly classified.

Class 2 (Suspicious): Significant misclassifications into class 0.

This indicates that the Decision Tree, while efficient, has limitations in handling certain complex phishing patterns.

### 6.2.2.Extra Trees Classifier Performance

The Extra Trees Classifier slightly outperformed the Decision Tree, achieving a test accuracy of 91.46%.



```
##############################################
######-Model => <class 'sklearn.ensemble._forest.ExtraTreesClassifier'>
Test Accuracy :  91.46%
              Classification_report
              precision    recall  f1-score   support

           0       0.92      0.98      0.95     85565
           1       0.93      0.97      0.95     19319
           2       0.83      0.57      0.68     18805
           3       0.97      0.91      0.94      6550

    accuracy                           0.91    130239
   macro avg       0.91      0.86      0.88    130239
weighted avg       0.91      0.91      0.91    130239

             Confusion_matrix
```

```
###################- End -###################
```

FIGURE 6.2 – EXTRA TREE CLASSIFIER REPORT

From the classification report:

Classes 0, 1, and 3 were well recognized with precision and recall values above 0.90. Class 2 still suffers from lower recall (0.57), though slightly better than the Decision Tree.

The confusion matrix provides the following insight:

Slight improvement in correctly identifying class 2 compared to the Decision Tree.

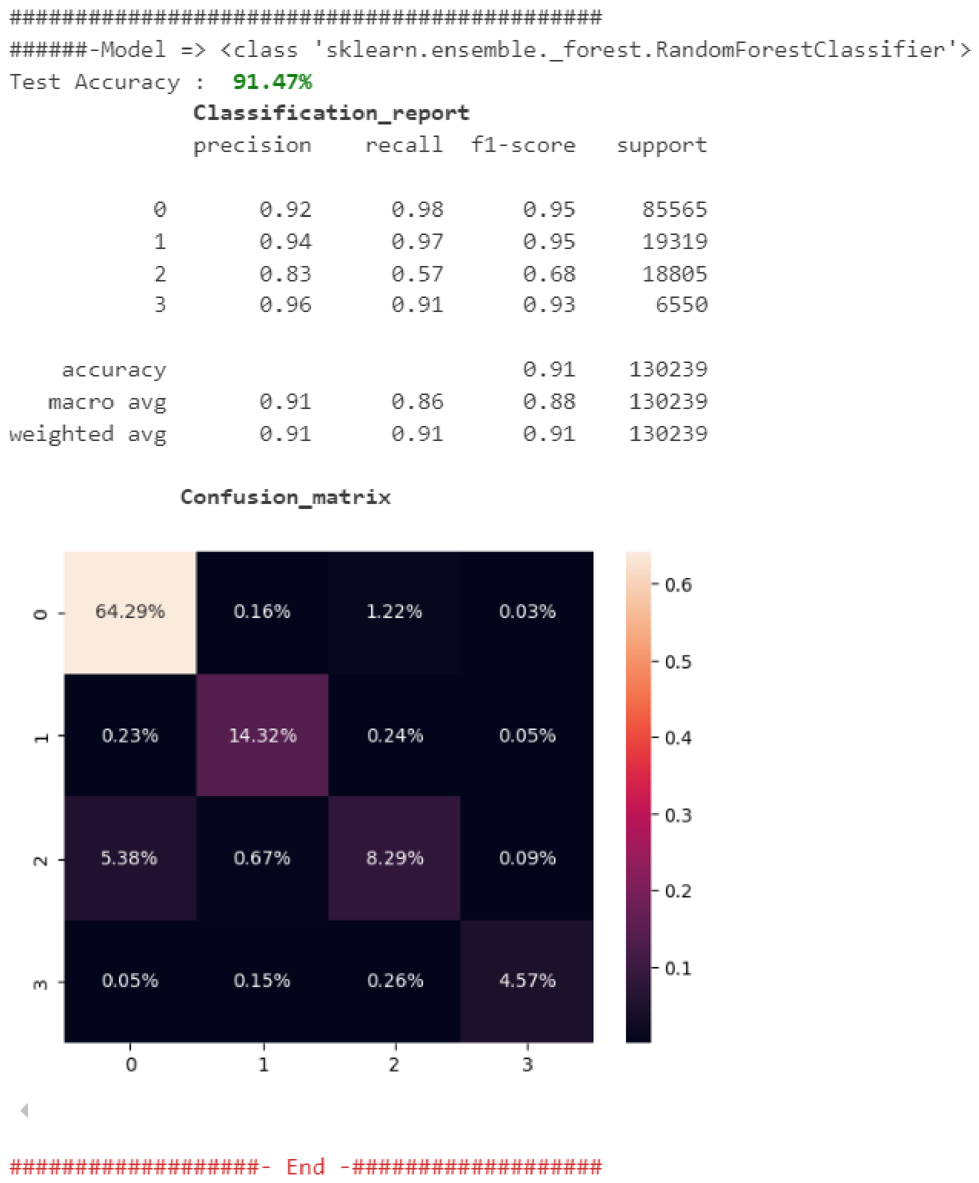
Reduced false positives in classes 0 and 1.

Visual clarity in the heatmap shows a more balanced classification across all four classes.

This model demonstrates enhanced generalization and robustness due to its ensemble nature, which aggregates multiple de-correlated trees.

## 6.2.3.Random Forest Classifier Performance

The Random Forest Classifier achieved the highest test accuracy of 91.47%, making it the best performer among the three.



```
#############################################
######-Model => <class 'sklearn.ensemble._forest.RandomForestClassifier'>
Test Accuracy :  91.47%
              Classification_report
              precision    recall  f1-score   support

           0       0.92      0.98      0.95     85565
           1       0.94      0.97      0.95     19319
           2       0.83      0.57      0.68     18805
           3       0.96      0.91      0.93      6550

    accuracy                           0.91    130239
   macro avg       0.91      0.86      0.88    130239
weighted avg       0.91      0.91      0.91    130239

             Confusion_matrix
```

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 64.29% | 0.16% | 1.22% | 0.03% |
| 1 | 0.23% | 14.32% | 0.24% | 0.05% |
| 2 | 5.38% | 0.67% | 8.29% | 0.09% |
| 3 | 0.05% | 0.15% | 0.26% | 4.57% |

```
##################- End -##################
```

FIGURE 6.3 – RANDOM FOREST CLASSIFIER REPORT

The classification report is nearly identical to that of Extra Trees, with slightly better values in precision and recall for class 2.

From the confusion matrix:

Class 0 and 1 predictions remain consistently accurate.

Class 2 recognition shows marginal improvement, minimizing its confusion with class 0.

Class 3 (likely a rare or critical type) was consistently predicted with high confidence across all classifiers.

Random Forest’s bagging strategy and randomness in feature selection significantly contribute to reducing overfitting and improving recall in difficult classes.

**6.3.Summary of outcomes**

The performance evaluation of the proposed real-time phishing detection system using ensemble learning techniques yielded promising outcomes across all three classifiers: Decision Tree, Extra Trees, and Random Forest. The following key points summarize the results:

High Overall Accuracy: All models achieved over 90% test accuracy, demonstrating reliable phishing detection capabilities even on unseen URLs.

Random Forest Classifier emerged as the best-performing model, achieving the highest accuracy of 91.47%, along with robust precision and recall metrics across all classes.

Extra Trees Classifier closely followed with 91.46% accuracy, showcasing excellent generalization and a slight improvement in handling complex or ambiguous phishing instances compared to the Decision Tree.

The Decision Tree Classifier, while achieving a respectable 90.94% accuracy, showed slightly lower performance in detecting certain classes, particularly class 2, indicating sensitivity to class imbalances or borderline features. Evaluation Metrics such as precision, recall, and F1-score were used to highlight not just accuracy but also the model’s ability to reduce false positives and false negatives. These metrics confirmed the system’s practical usability in real-time threat environments.

Confusion Matrices helped visually assess the misclassification trends, guiding future enhancements such as data augmentation, rebalancing strategies, or additional feature engineering to improve class-specific accuracy.

In conclusion, the ensemble learning-based approach used in this system successfully achieved reliable, real-time phishing detection. The models performed exceptionally well in identifying and classifying different URL types, with potential for further improvements in complex or ambiguous cases. This lays a solid foundation for deploying the system in practical, user-facing environments where online safety and threat prevention are crucial.

# CHAPTER - 7

# CONCLUSION & FUTURE ENHANCEMENTS

The Real-Time Phishing Detection System developed in this work offers a sophisticated approach to mitigating the risks associated with phishing attacks. By leveraging machine learning, specifically an ensemble of Decision Tree, Random Forest, and Extra Trees classifiers, the system efficiently identifies and classifies URLs as either "Phishing" or "Benign."

This classification process relies on the extraction of specific URL features, such as length, structure, and the presence of special characters or digits, which are commonly associated with phishing attempts. The system's design and implementation aim to provide a solution that is both accurate and user-friendly, allowing it to serve as a practical tool for individuals, organizations, and web service providers seeking to enhance their cybersecurity measures.

The` API-based architecture further contributes to the system's accessibility by allowing easy integration into a variety of applications, such as web platforms, browser extensions, or mobile apps. This ensures that users can receive immediate feedback on the legitimacy of URLs they encounter in real-time, reducing the risk of falling victim to phishing scams.

## Outcomes and Achievements

### High Detection Accuracy and Reliability:

Through an ensemble approach combining multiple machine learning models, this system achieves high accuracy rates in detecting phishing URLs. The use of diverse classifiers improves reliability by reducing the likelihood of misclassification, ensuring that most phishing URLs are accurately flagged.

### Real-Time Processing Capabilities:

The system has been optimized for real-time performance, allowing for rapid prediction by using a streamlined feature extraction process. This makes the system suitable for real-world applications where immediate detection is crucial to prevent phishing attacks.

**User-Friendly Integration and Scalability:**

The implementation of the system as FastAPI allows it to be easily integrated into a range of client applications. This design choice not only enhances accessibility but also enables scaling up to meet the demands of larger user bases or enterprise environments with minimal adjustments.

## Future Enhancements

While the current system performs well in terms of detection accuracy and ease of integration, there are several areas where it could be enhanced to ensure greater robustness, adaptability, and user protection. These potential future improvements include:

**Behavioral Analysis for Enhanced Detection**:

Integrating behavioral analysis, such as tracking suspicious redirects, examining page content, or monitoring server response behaviors, could provide additional context that enhances detection accuracy. Behavioral patterns of phishing websites often differ significantly from legitimate sites, and capturing these can further reduce false positives and false negatives.

**Continuous Learning and Model Adaptation:**

As phishing tactics evolve, so must detection systems. A potential enhancement would be to incorporate continuous learning mechanisms, allowing the system to adapt to new phishing patterns by periodically retraining models with fresh data. This could be accomplished by integrating with a dynamic dataset that updates regularly with recent phishing URLs.

**Analysis of Embedded Links on Web Pages:** The current system only evaluates standalone URLs. In the real world, phishing content often hides within embedded links on a legitimate-looking webpage. Future versions of this system could parse the entire webpage after loading and analyze all internal anchor (<a>) tags and embedded scripts to extract and scan suspicious links in real time.

**Context-Aware Detection:** Integrating Natural Language Processing (NLP) to analyze webpage content such as login prompts, alert messages, and suspicious forms can enhance phishing detection.

**Enhanced Security and Privacy Measures:**

Since the system processes URLs that could potentially contain sensitive information, implementing strong data privacy practices, such as encryption and anonymization, is critical. Protecting user data will increase trust in the system, making it more attractive for widespread adoption in security-conscious environments.

**Deployment on Scalable Cloud Infrastructure**:

To meet the needs of larger user bases and enterprise applications, deploying the system on a cloud infrastructure, such as AWS or Google Cloud, would enhance scalability. Cloud deployment offers additional benefits, including automated load balancing, scalability to handle high traffic, and ease of maintenance. Additionally, cloud-based deployment can support serverless architectures, which could further optimize costs and performance for large-scale operations.

**Final Reflections**

The Real-Time Phishing Detection System presented in this work showcases the effectiveness of machine learning in addressing the persistent challenge of phishing attacks. By building a system that is not only accurate but also responsive and accessible, we have contributed a practical tool for improving cybersecurity.

This tool offers significant benefits for a wide range of users, from individual internet users looking for safer browsing to enterprises and web service providers aiming to protect their customers. The integration of machine learning in cybersecurity highlights the vast potential of technology to combat online threats in a proactive manner. As cyber threats evolve, the need for adaptive, intelligent, and scalable security solutions will only grow.

The insights gained from this work provide a foundation for future advancements, laying the groundwork for enhanced protection against phishing and potentially other forms of cyber-attacks. This work has the potential to evolve further, integrating cutting-edge technologies and adapting to new threat landscapes. With continuous improvements, the Real-Time Phishing Detection System could become a comprehensive cybersecurity solution that not only detects phishing attacks but also actively contributes to a safer internet environment for all users.

## REFERENCE

[1] Vegesna, V.V., (2023). Enhancing Cyber Resilience by Integrating AI-Driven Threat Detection and Mitigation Strategies. Transactions on Latest Trends in Artificial Intelligence, 4(4).

[2] Bonfanti, M.E., (2022). Artificial intelligence and the offence-defence balance in cyber security. Cyber Security: Socio-Technological Uncertainty and Political Fragmentation. London: Routledge, pp.64-79.

[3] Vegesna, V.V., (2023). Comprehensive Analysis of AI-Enhanced Defense Systems in Cyberspace. International Numeric Journal of Machine Learning and Robots, 7(7).

[4] Guembe, B., Azeta, A., Misra, S., Osamor, V.C., Fernandez-Sanz, L. and Pospelova, V., (2022). The Emerging Threat of Ai-driven Cyber Attacks: A.

[5] Zeadally, S., Adi, E., Baig, Z. and Khan, I.A., (2020). Harnessing artificial intelligence capabilities to improve cybersecurity. Ieee Access, 8, pp.23817-23837.

[6] Sjöblom, C., (2021). Artificial Intelligence in Cybersecurity and Network security.

[7] Kaloudi, N. and Li, J., (2020). The ai-based cyber threat landscape: A survey. ACM Computing Surveys (CSUR), 53(1), pp.1-34.

[8] Guembe, B., Azeta, A., Misra, S., Osamor, V.C., Fernandez-Sanz, L. and Pospelova, V., (2022). The Emerging Threat of Ai-driven Cyber Attacks: A.

[9] Bokhari, S.A.A. and Myeong, S., (2023). The influence of artificial intelligence on e-Governance and cybersecurity in smart cities: A stakeholder's perspective. IEEE Access.

[10] Kumar, S., Gupta, U., Singh, A.K. and Singh, A.K., (2023). Artificial intelligence: revolutionizing cyber security in the digital era. Journal of Computers, Mechanical and Management, 2(3), pp.31-42.

[11] Zeadally, S., Adi, E., Baig, Z. and Khan, I.A., (2020). Harnessing artificial intelligence capabilities to improve cybersecurity. Ieee Access, 8, pp.23817-23837.

[12] Kaloudi, N. and Li, J., (2020). The ai-based cyber threat landscape: A survey. ACM Computing Surveys (CSUR), 53(1), pp.1-34.

[13] Rangaraju, S., (2023). AI SENTRY: REINVENTING CYBERSECURITY THROUGH INTELLIGENT THREAT DETECTION. EPH-International Journal of Science And Engineering, 9(3), pp.30-35.

[14] Soni, V.D., (2020). Challenges and Solution for Artificial Intelligence in Cybersecurity of the USA. Available at SSRN 3624487.

[15] Markevych, M. and Dawson, M., (2023). A review of enhancing intrusion detection systems for cybersecurity using artificial intelligence (ai). In International conference KNOWLEDGE-BASED ORGANIZATION (Vol. 29, No. 3, pp. 30-37).