**ADHIYAMAAN COLLEGE OF ENGINEERING**

**(An Autonomous)**

**Dr. MGR Nagar – Hosur**

**Regulation – 2024**

**Batch 2024-2026**

**Semester – III**

**Mini Project**

**First Review – Details**

**25th July 2025**

**DEEP DETECT**

**SMART PHISHING URL DETECTION WEB APPLICATION**

**Technology Used:**

**HTML, CSS, JavaScript , PHP , MYSQL**

**DHIVAGHAR.M**

**DEPARTMENT OF MCA**

**2403617662221008**

**Problem Statement**

As millions of users rely on websites for banking, shopping, and communication, phishing attacks have become one of the most prevalent and dangerous cybersecurity threats. Attackers craft fake websites that appear legitimate in order to steal sensitive information like login credentials, credit card numbers, and personal data.

Traditional solutions—such as browser-based warnings and static blacklists—are limited in their ability to detect newly generated, cleverly disguised phishing URLs. These outdated methods cannot scale with the speed and complexity of modern phishing attacks, leaving users vulnerable.

Deep Detect addresses this issue by offering a smart, machine learning-based web application that allows users to enter a URL and instantly receive a prediction on whether the link is safe or phishing. By analyzing multiple structural and behavioral characteristics of URLs, the system provides accurate, real-time classification and helps users make safer browsing decisions.

This solution not only enhances user protection but also demonstrates the power of combining AI, cyber security, and full-stack web development in a meaningful, realworld application.

|  |  |
| --- | --- |
| **Issue** | **Impact** |
| No centralized phishing detection tool | Users rely on outdated blacklists or browser warnings that miss new threats |
| Complex or technical tools | Non-technical users struggle to verify website safety |
| Lack of real-time URL analysis | Users can fall victim before the threat is identified |
| No intelligent pattern recognition | Traditional methods fail to detect evolving or obfuscated phishing links |
| No easy-to-use safety checker | Users click harmful links without verifying their legitimacy |

**Why Is This Problem Important?**

# Project Objectives – SMART Format

## Specific

Build a secure, intelligent web application that allows users to submit URLs and instantly receive a phishing safety status using machine learning. The system will extract relevant URL features, classify the input using a trained model, and provide clear results — while optionally logging the scan for future analysis.

## Measurable

* 100% of submitted URLs will be processed and logged with unique IDs (if logging enabled)
* Results (Safe/Phishing) will be displayed in < 3 seconds after submission
* Achieve 90%+ prediction accuracy on phishing classification
* Allow batch testing of up to 10 URLs at a time (optional extension)

## Achievable

Developed using Flask (Python) for backend, HTML/CSS/JavaScript for frontend, and scikit-learn for ML, the system will follow modular architecture, support URL feature extraction, prediction, result display, and optional admin-level scan tracking.

## Relevant

As phishing remains one of the top cyber threats worldwide, Deep Detect empowers users with a real-time tool to verify link safety. It also demonstrates the practical use of **AI and cybersecurity** in solving real-world challenges — making it highly relevant for MCA students and the digital age.

## Time-bound

* **Week 1–2**: Requirement gathering, UI design, ML dataset collection
* **Week 3–4**: Feature extraction and ML model development
* **Week 5–6**: Flask API, frontend integration, and result rendering
* **Week 7–8**: Optional database logging, result history module
* **Week 9–10**: Final testing, optimization, documentation, and deployment

# Software Requirements Specification (SRS)

## Functional Requirements

### 1)User Functions (General Users)

* **Enter URL for Verification**

Users can input any website URL through the web interface for phishing detection.

* **View Prediction Result**

After processing, users see a clear result: **Safe**  or **Phishing**  .

* **Scan History (Optional – if logged in)**

Logged-in users can view their previous URL checks along with timestamps and results.

* **Batch URL Testing (Optional Feature)**

Users can upload or enter multiple URLs for simultaneous scanning.

* **Register/Login (Optional)**

Users may register to access extra features like scan history or dashboard personalization.

### 2 )Admin Functions (Optional – Advanced Feature)

* **Admin Login**

Secure login for admin access to manage logs and monitor scanning activity.

* **View All Scan Logs**

Admins can view a dashboard showing user-submitted URLs, prediction results, and timestamps.

* **Manage ML Model**

Admins can view details of the currently deployed ML model and optionally upload a new trained model file.

* **Moderate User Activity**

Admins can monitor or flag misuse, such as repeated scans of known phishing links.

## Non-Functional Requirements

### 1. Security

* Sanitize all URL inputs to prevent injection attacks.
* HTTPS must be enforced for all deployments.
* If user accounts are used, passwords must be hashed using bcrypt or a similar method.
* Role-based access control ensures only admins can access moderation/log management tools.
* Basic protection against XSS, CSRF, and other common web vulnerabilities.

### 2. Performance

* URL scan and result display should occur within **3 seconds** in 95% of cases.
* Batch URL checks (up to 10 URLs) should complete within **5 seconds**.
* Capable of supporting **100+ concurrent users** on standard cloud hosting.

### 3. Usability

* The UI must be clean, intuitive, and responsive on mobile and desktop.
* Results should be color-coded and clearly indicate the classification.
* Tooltips or explanations should help users understand what phishing means.

### 4. Reliability

* Application uptime must be ≥ 99% during active hours.
* Scans should not be lost or timed out unexpectedly.
* Logs (if enabled) must persist through system refresh or downtime.

**5. Scalability**

* Modular structure allows easy upgrades to add more features like browser extension integration or user accounts.
* Database and server infrastructure (e.g., Render, PythonAnywhere, Firebase) must scale horizontally if needed.

### 6. Maintainability

* Code will follow modular MVC structure (especially Flask-based).
* Clear comments and documentation throughout the backend and frontend.
* ML model stored as a .pkl or .joblib file for easy updates and retraining.
* Easily upgradable to integrate different ML algorithms (e.g., SVM, Neural Nets).

### 7. Auditability

* Every URL scan (if logging is active) will be timestamped and stored.
* Admin panel (if implemented) will include filters and search to audit system usage.
* Logs include: URL, result, user (if logged in), time, and status.

# User Characteristics

* **General Users:** 
  + Typically non-technical individuals who want to check whether a URL is safe. o Comfortable with simple web forms (copy/paste URL and click submit).
  + Expect quick, clear feedback on whether a site is safe or phishing.
* **Admin Users (Optional):** 
  + Semi-technical users who monitor usage logs, model status, and application health.
  + Can handle model updates or manage scan histories and misuse tracking.
  + Access admin dashboard for insights or moderation tasks (if implemented).

# Operating Environment

|  |  |
| --- | --- |
| **Layer** | **Technology Used** |
| **Frontend** | HTML5, CSS3, JavaScript (Vanilla JS or Bootstrap) |
| **Backend** | Python with Flask Framework |
| **Machine Learning** | scikit-learn, pandas, joblib (using Logistic Regression / Random Forest algorithms) |
| **Database (Optional)** | SQLite / MySQL – for storing URL scan logs |
| **Hosting / Deployment** | Render, PythonAnywhere, or localhost (can use XAMPP for testing) |
| **Devices** | Desktop or Mobile browser – for scanning URLs and accessing admin portal (if any) |

# System Diagrams

The following diagrams are included for Deep Detect:

* **Context Level DFD** – Shows system interaction with users.
* **Level 1 & Level 2 DFDs** – Breaks down internal processes like input, prediction, and result handling.
* **Use Case Diagram** – Visualizes user actions like submit URL and view result.
* **Activity Diagram** – Describes step-by-step flow of phishing detection.
* **Class Diagram** – Displays main system classes and their relationships.
* **ER Diagram** – Shows database structure (Users, URLs, Results).

# Database Design (ER Model)

## 1. Users

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| user\_id | INT (PK) | Unique user ID |
| name | VARCHAR | User’s full name |
| email | VARCHAR | User’s email (unique) |
| password\_hash | VARCHAR | Hashed password (if login enabled) |
| created\_at | DATETIME | Registration date |

## Scans

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| scan\_id | INT (PK) | Unique scan ID |
| user\_id | INT (FK) | Linked to Users table (nullable for guest) |
| url | TEXT | URL submitted by user |
| prediction | VARCHAR | Result: Safe or Phishing |
| confidence | FLOAT | Confidence score from ML model |
| scanned\_at | DATETIME | Timestamp of scan |

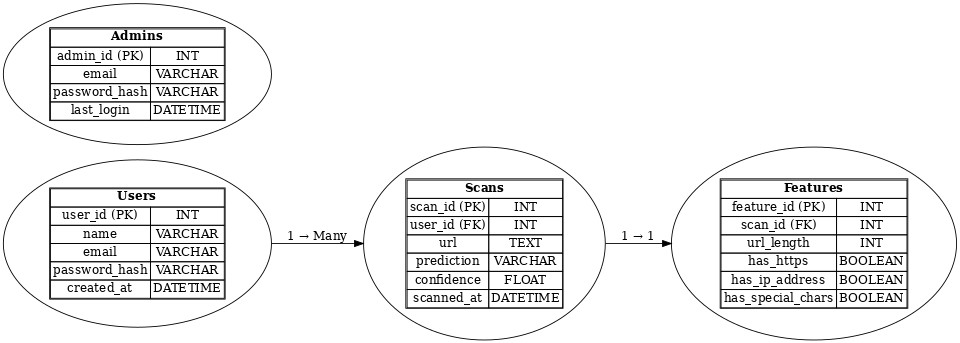
## Features (Optional – if logging extracted features)

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| feature\_id | INT (PK) | Unique feature ID |
| scan\_id | INT (FK) | Linked to Scans |
| url\_length | INT | Length of the URL |
| has\_https | BOOLEAN | If the URL uses HTTPS |
| has\_ip\_address | BOOLEAN | If URL contains IP address |
| has\_special\_chars | BOOLEAN | Special characters in the URL |
| ... | ... | (Any other features you extract) |

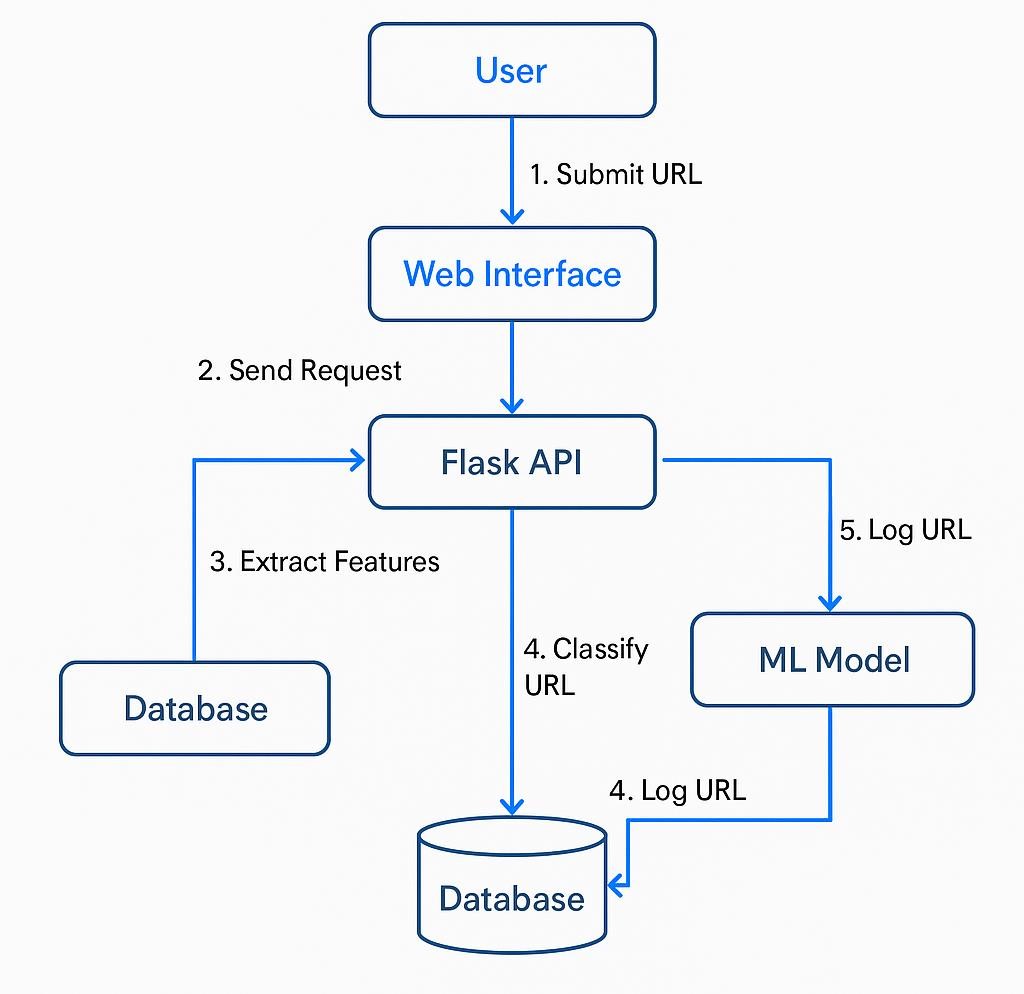
## Admins (Optional)

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| admin\_id | INT (PK) | Unique admin ID |
| email | VARCHAR | Admin login |
| password\_hash | VARCHAR | Hashed password |
| last\_login | DATETIME | Last login timestamp |

Database diagram



Data flow diagram



ER diagram

