@GittyCandy

Ahsan Latif

@GittyCandy

AI Powered Phishing Detector



Table of Contents

[**Introduction** 3](#_Toc181974188)

[**Overview** 4](#_Toc181974189)

[**Model Evaluation** 5](#_Toc181974190)

[**Performance Metrics Used** 5](#_Toc181974191)

[**Cross Validation between multiple models** 5](#_Toc181974192)

[**Result Analysis:** 6](#_Toc181974193)

[**Why did we choose Logistic Regression over the other models?** 6](#_Toc181974194)

[**Code Explanation** 7](#_Toc181974195)

[**Web Application** 8](#_Toc181974196)

[**User Interface** 8](#_Toc181974197)

[**Analysis Report** 11](#_Toc181974198)

[**Analysis.txt File for the web Page** 11](#_Toc181974199)

[**Steps to run the Webpage** 12](#_Toc181974200)

[**Conclusion** 13](#_Toc181974201)

[**References** 13](#_Toc181974202)

[**Work Contribution** 14](#_Toc181974203)

# **Introduction**

Phishing attacks are a growing threat in today’s digital world, where malicious URLs trick users into revealing sensitive information. These attacks can cause severe financial and reputational damage to both individuals and businesses. With phishing becoming more sophisticated, it’s crucial to have tools that can quickly identify fake URLs and protect users from falling victim to these scams.

Our goal with this project was to create a Phishing URL Detection System that not only flags fake URLs but also provides a detailed explanation of why a URL was classified as legitimate or suspicious. Using machine learning, we developed a system that analyzes the features of a URL and predicts whether it’s safe or malicious.

We chose **Logistic Regression** as the model for this project due to its high accuracy and reliability in classifying phishing URLs. The system is built into a **Node.js web application**, which allows users to input a URL and receive a prediction along with an **analysis.txt** file. This file provides insights into the model's decision-making process, offering transparency and helping users understand what factors contributed to the classification.

here, we’ll discuss the development process, the models we evaluated, and why we ultimately selected Logistic Regression. We’ll also explain how the system works, how to run it, and how it helps users stay protected from phishing attacks.

# **Overview**

In this project, we developed a sophisticated system aimed at detecting whether a given URL is legitimate or fake using machine learning models. Our approach involved training multiple models on a phishing dataset, where each model was carefully evaluated to determine its effectiveness in distinguishing between legitimate and phishing URLs. The models included various algorithms, such as Logistic Regression, K-Nearest Neighbors, Support Vector Machine, Naive Bayes Classifier, Decision Tree, and others. After comparing their performance based on metrics like accuracy, F1 score, recall, and precision, the Logistic Regression model emerged as the best performer, achieving the highest accuracy and reliability. This made it the ideal choice for deployment in our application.

The selected Logistic Regression model was then integrated into a web application built with Node.js, allowing users to easily input a URL into the system. Once the user submits a URL, the system predicts whether the URL is legitimate or fake, providing a clear and accurate response. Additionally, users can download an analysis report (in .txt format), which outlines the backend processes and explains the feature extraction and classification steps that led to the model's decision.

This approach not only makes phishing detection accessible to users but also provides transparency regarding how the system works behind the scenes. For those interested in understanding the implementation in more detail, the full explanation of the code and its functionality is provided in the accompanying Jupyter notebook file.

## **Model Evaluation**

### **Performance Metrics Used**

We evaluated the following machine learning models:

* **Logistic Regression**
* **K-Nearest Neighbors (KNN)**
* **Support Vector Machine (SVM)**
* **Naive Bayes**
* **Decision Tree**
* **Random Forest**

| **ML Model** | **Accuracy** | **F1 Score** | **Recall** | **Precision** |
| --- | --- | --- | --- | --- |
| Logistic Regression | 0.934 | 0.941 | 0.943 | 0.927 |
| K-Nearest Neighbors | 0.956 | 0.961 | 0.991 | 0.989 |
| Support Vector Machine | 0.964 | 0.968 | 0.980 | 0.965 |
| Naive Bayes | 0.605 | 0.454 | 0.292 | 0.997 |
| Decision Tree | 0.958 | 0.962 | 0.991 | 0.993 |
| Random Forest | 0.967 | 0.971 | 0.995 | 0.988 |
|  |  |  |  |  |

Each model was assessed using four key performance metrics: accuracy, F1 score, recall, and precision. The table below summarizes the results:

### **Cross Validation between multiple models**

### **Result Analysis:**

After evaluating the performance of all models, **Logistic Regression** was chosen as the best model for this project. Although models like K-Nearest Neighbors (KNN) and Random Forest showed high performance, Logistic Regression provided a balanced combination of accuracy and interpretability, which is crucial for a project that involves understanding and explaining predictions. Logistic Regression's performance was consistently strong across all evaluation metrics, with an accuracy of 93.4%, an F1 score of 94.1%, and a recall of 94.3%, making it the ideal choice for this phishing detection system.

The Logistic Regression model's ability to deliver high accuracy, alongside its interpretability and relatively straightforward implementation, outweighed the small performance improvements observed in other models like KNN and Random Forest.

For further technical details, including code implementation and model evaluation, please refer to the accompanying Jupyter notebook.

### **Why did we choose Logistic Regression over the other models?**

While many models showed strong performance, **Logistic Regression** was selected for its excellent balance of **accuracy** (0.934), **F1 score** (0.941), and **recall** (0.943) on both training and testing datasets. Compared to other models like K-Nearest Neighbors and Support Vector Machine, Logistic Regression offered the most consistent performance across different metrics, making it the optimal choice for the project.

**Advantages of Logistic Regression over Other Models**

* **Simplicity and Interpretability**: Logistic Regression is a simple linear model, which makes it easy to understand and interpret, especially compared to more complex models like Support Vector Machine and Random Forest.
* **Efficient and Fast**: Logistic Regression is computationally efficient and relatively quick to train, making it ideal for real-time applications like URL prediction.
* **Stable Performance**: Despite the potential for more complex models to outperform Logistic Regression in certain cases, it provides stable and reliable predictions, especially in the context of phishing URL detection, where features are often linear.
* **Less Overfitting**: Logistic Regression is less prone to overfitting compared to models like Decision Trees or Random Forest, which can have high variance, especially when the training data is limited.

## **Code Explanation**

Each part of the code is explained in detail in the Jupyter file, ensuring clarity and understanding of how the system functions.

# **Web Application**

### **User Interface**

Using a legitimate URL, from ChatGPT:

A screenshot of a computer

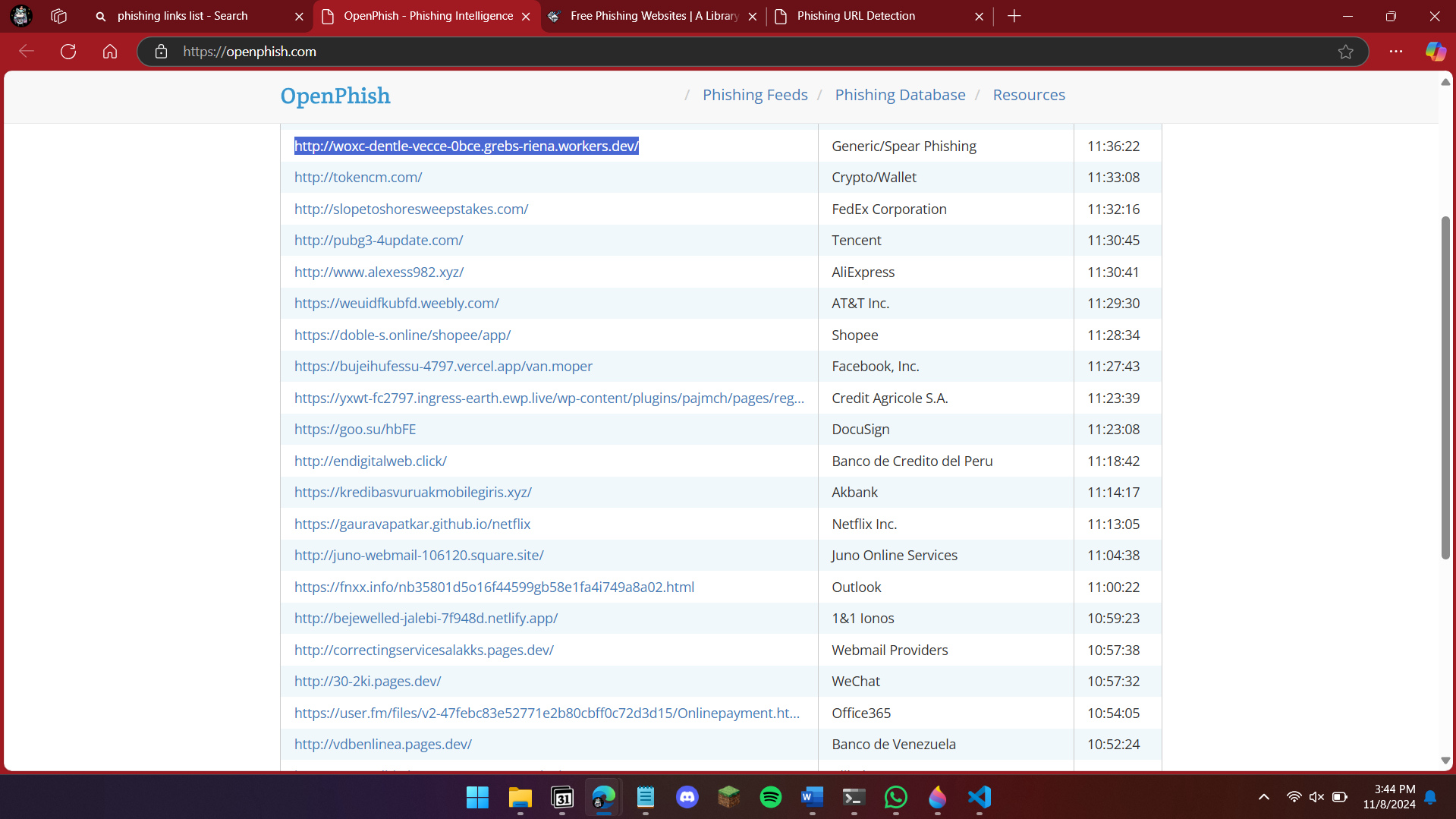
Description automatically generated

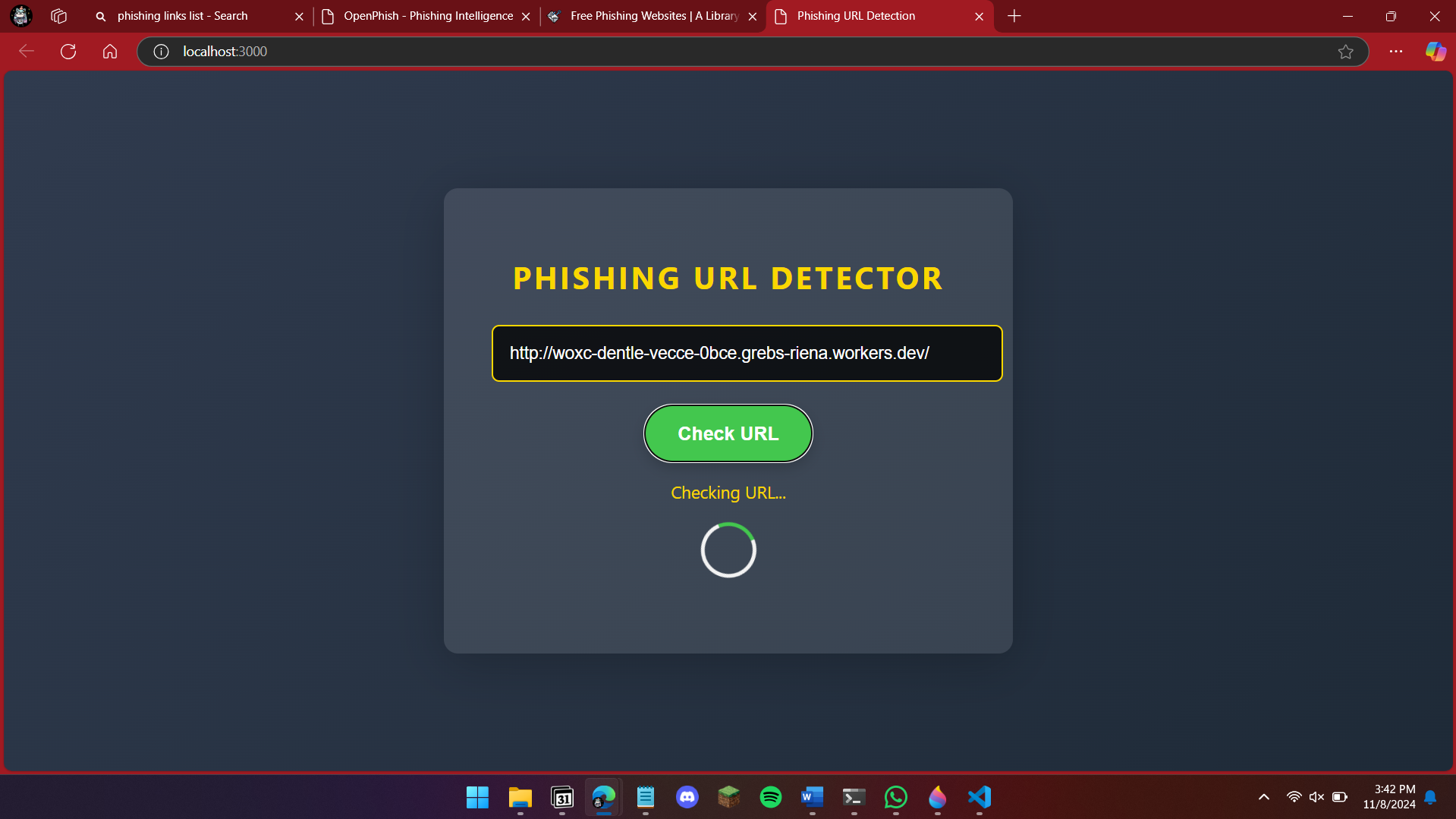
Analysis:

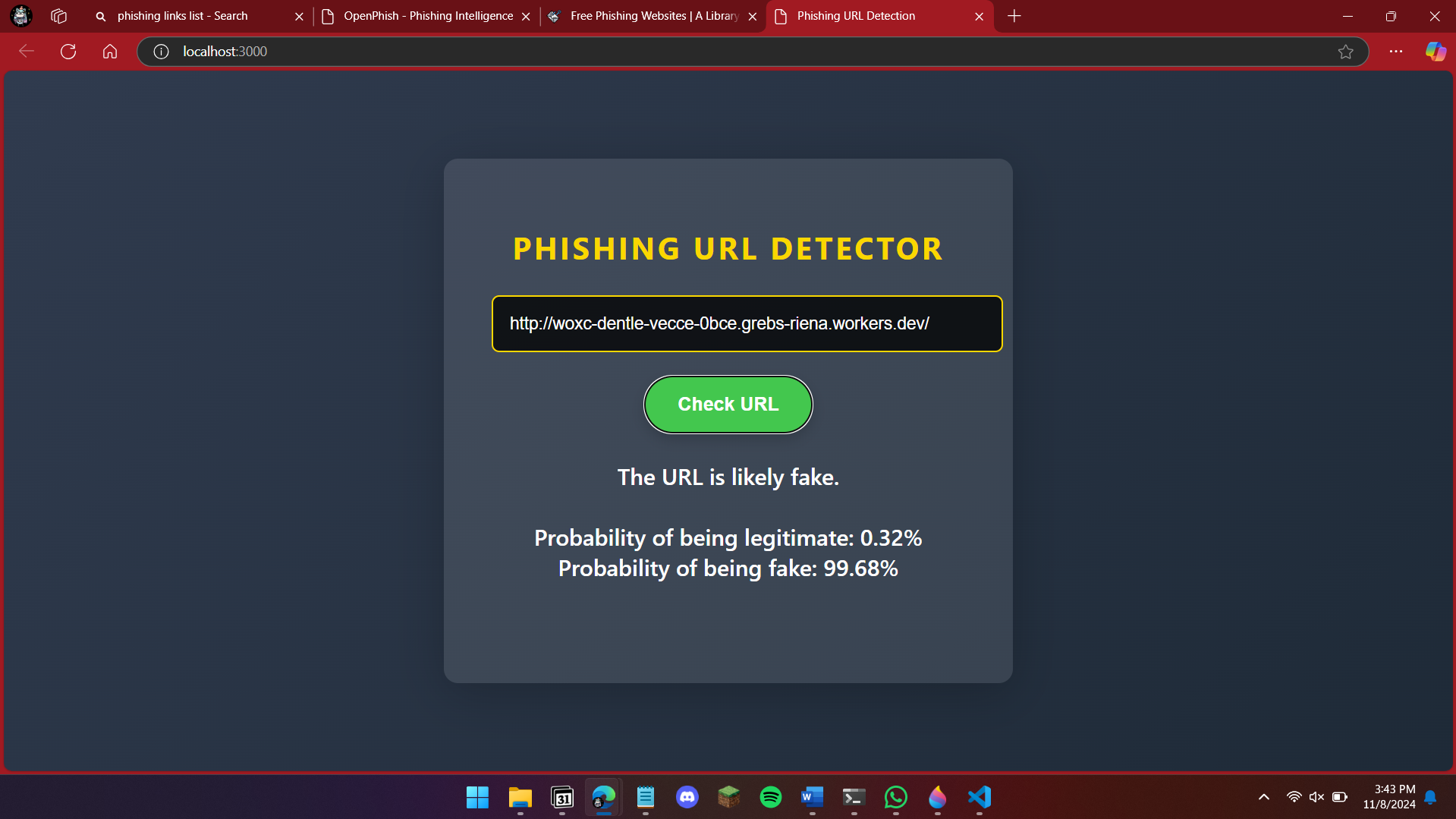
A screen shot of a computer

Description automatically generated

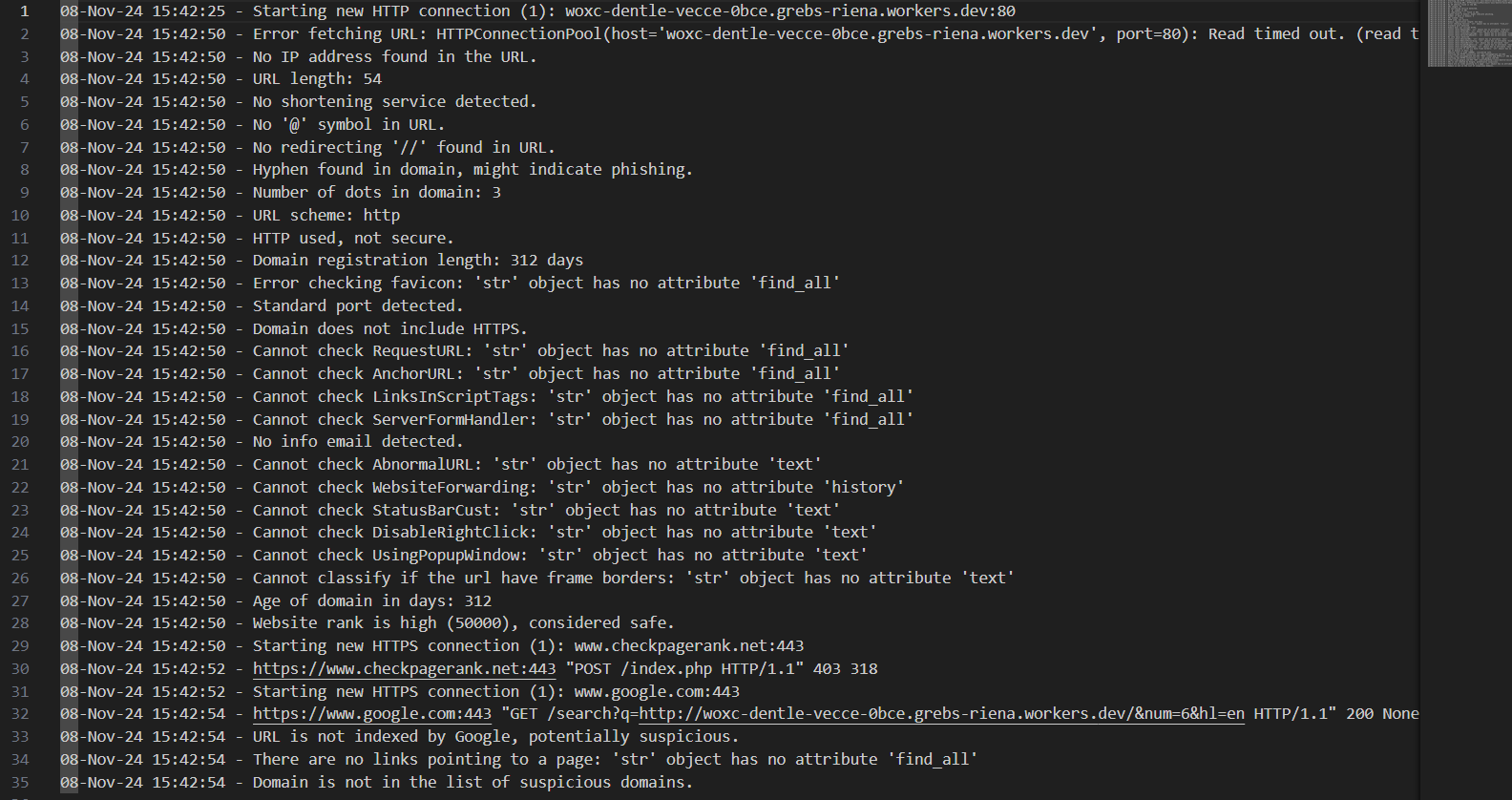
Using a General Spear Phishing URL, from OpenPhish (website with the latest phishing link):







Analysis:

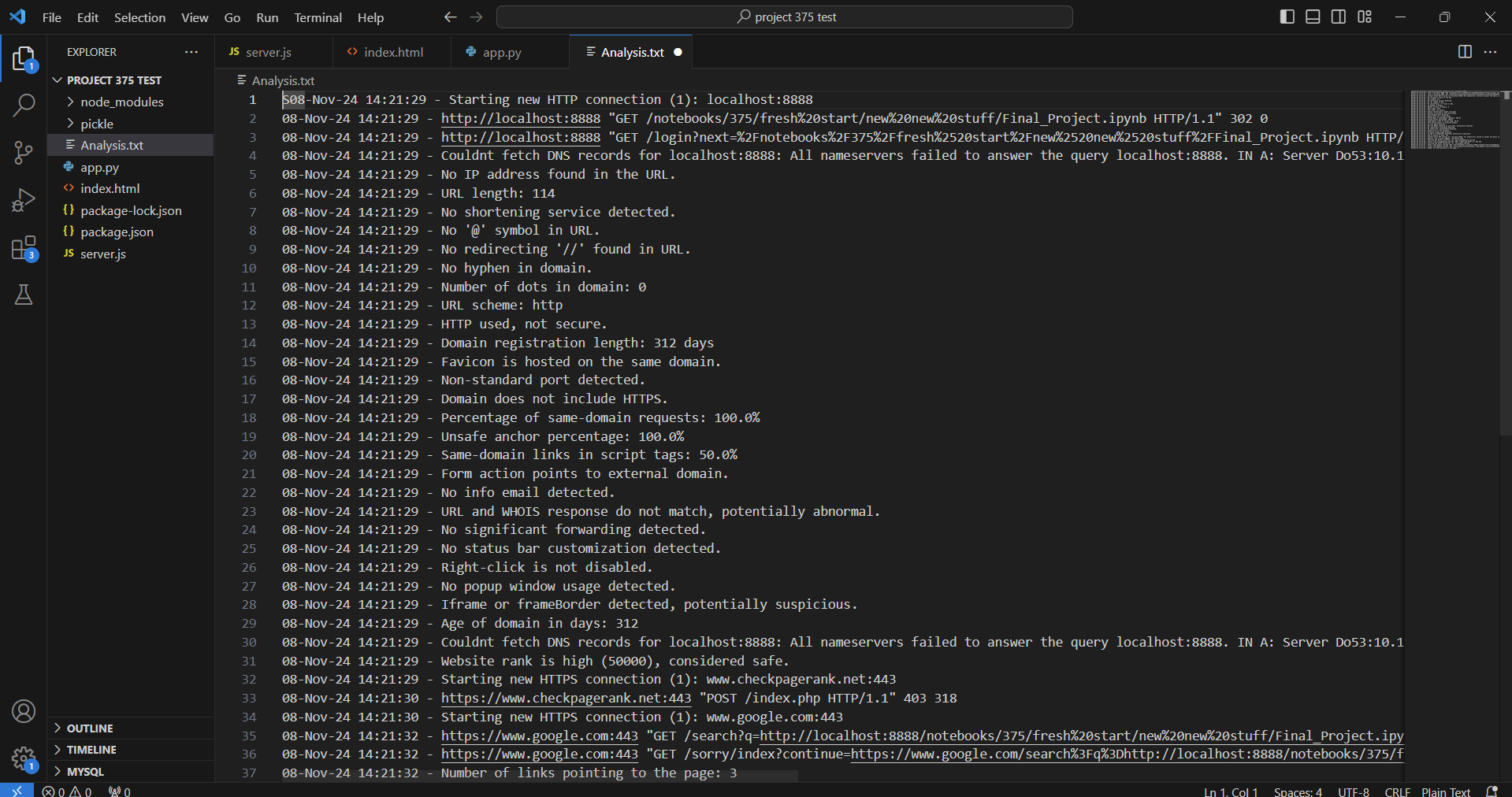


To make the system accessible to users, we developed a web application using **Node.js**. The application allows users to input a URL, which is then analyzed by the Logistic Regression model. After processing, the system displays whether the URL is **legitimate** or **fake**.

### **Analysis Report**

In addition to the prediction, users can download an **analysis.txt** file. This report provides an in-depth look at the extracted features of the URL and the prediction process. It includes the model's decision-making process, allowing users to see exactly what factors contributed to the result. This transparency is crucial for users who want to understand how the system works behind the scenes and ensures confidence in the predictions.

### **Analysis.txt File for the web Page**



The **analysis.txt** file is a key part of this project as it provides in-depth details on what’s happening behind the scenes. This file explains:

* The features being extracted from the URL
* The steps taken during URL analysis
* Predictions made by the machine learning model
* Confidence levels (probabilities) for both legitimate and fake outcomes

This analysis is valuable for users who want to understand the reasoning behind the URL classification and the model's performance.

### **Steps to run the Webpage**

To get started with the application, follow these steps:

**Step 1: Install Required Libraries**

Before running the application, you need to install the necessary libraries.

1. **Install Node.js Libraries:**

First, ensure that **Node.js** is installed on your system. Then, install the required Node.js libraries:

npm install requests

npm install cors

1. **Install Python Libraries:**

Since we’re using Python for feature extraction and machine learning, you’ll also need to install some Python libraries:

pip install requests beautifulsoup4 googlesearch-python dnspython

**Step 2: Run the Application**

Once the libraries are installed, you can run the web application, which will allow users to input a URL and get predictions along with a detailed analysis of the results.

# **Conclusion**

The phishing URL detection system is an efficient and reliable tool for identifying malicious URLs. The combination of machine learning and a simple user interface ensures that even non-technical users can quickly check the legitimacy of URLs. The addition of the **analysis.txt** file is an essential feature that provides transparency, showing users the inner workings of the system’s decision-making process.

By selecting Logistic Regression and integrating it into a Node.js web application, we’ve created a tool that not only performs well but is easy to use and understand.

# **References**

**Kaggle Dataset**  
Shashwatwork. (2021). *Web Page Phishing Detection Dataset*. Kaggle. Retrieved from <https://www.kaggle.com/datasets/shashwatwork/web-page-phishing-detection-dataset>

**YouTube Video**  
Practical Deep Learning. (2021). *Phishing Detection Techniques*. YouTube. Retrieved from <https://www.youtube.com/watch?v=8dF_NGBqJ-0>

**Google Search Python Library Documentation**  
Python-googlesearch. (n.d.). *Google Search Python Library*. Retrieved from <https://pypi.org/project/googlesearch-python/>

**BeautifulSoup Documentation**  
BeautifulSoup. (n.d.). *BeautifulSoup: Python Library for Parsing HTML and XML*. Retrieved from https://www.crummy.com/software/BeautifulSoup/

**Logistic Regression Model Explanation**  
OpenAI. (2024). *How Logistic Regression Works: ChatGPT Explanation*. Retrieved from <https://openai.com/>

**Github For Inspiration**

[Yukesh1004/Phising-Link-Detector-using-Flask-ML: Phishing attacks exploit user trust with fake websites mimicking real ones, risking data breaches and financial loss. This project aims to combat this threat using machine learning (ML) and artificial intelligence (AI) to identify phishing domains. By analyzing URL features, the system detects and prevents fraud effectively.](https://github.com/Yukesh1004/Phising-Link-Detector-using-Flask-ML)

This GitHub project used a rigid dataset with 3 features, but was a great starting point