Phishing Website/URL  
Detection

***: Low Level Design (LLD) :***



CREATED BY

1. **Priya Karvande**
2. **Jayesh Patil**
3. **Nikhil Dahitule**

## Document Version Control

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Description** |
| 28-Feb-2023 | 1.0 | Initial LLD |
|  |  |  |
|  |  |  |
|  |  |  |

Table of Contents

[Document version control 2](#_TOC_250015)

1. [Introduction 4](#_TOC_250014)
   1. [Why this Low-level design Document ? **4**](#_TOC_250013)
   2. [Scope **4**](#_TOC_250012)
2. [Architecture 5](#_TOC_250011)
3. [Architecture Description 6](#_TOC_250010)
   1. [Data for the problem **6**](#_TOC_250009)
   2. [Data Transformation **6**](#_TOC_250008)
   3. [Data insertion into database **6**](#_TOC_250007)
   4. [Export data from database **6**](#_TOC_250006)
   5. [Data pre-processing **6**](#_TOC_250005)
   6. [Model building **6**](#_TOC_250004)
   7. [Data from user **6**](#_TOC_250003)
   8. Data pre-processing **8**
   9. Model calling for data **7**
   10. [Predicted data **7**](#_TOC_250002)
   11. [Saving output at database **7**](#_TOC_250001)
4. [Unit Test Cases 8](#_TOC_250000)

# Introduction:

### Why this Low-Level Design Document?

The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code for Credit card default prediction System. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document.

### Scope

Low-level design (LLD) is a component-level design process that follows a step-by-step refinement process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

# Architecture

# 

# Architecture Description

### Data for the problem

Malicious URLs or malicious website is a very serious threat to cybersecurity. Malicious URLs host unsolicited content (spam, phishing, drive-by downloads, etc.) and lure unsuspecting users to become victims of scams (monetary loss, theft of private information, and malware installation), and cause losses of billions of dollars every year. We have collected this dataset to include a large number of examples of Malicious URLs so that a machine learning-based model can be developed to identify malicious urls so that we can stop them in advance before infecting computer system or spreading through internet.

we have collected a huge dataset of 651,191 URLs, out of which 428103 benign or safe URLs, 96457 defacement URLs, 94111 phishing URLs, and 32520 malware URLs. Figure 2 depicts their distribution in terms of percentage. As we know one of the most crucial tasks is to curate the dataset for a machine learning project. We have curated this dataset from five different sources.

For collecting benign, phishing, malware and defacement URLs we have used URL dataset (ISCX-URL-2016) For increasing phishing and malware URLs, we have used Malware domain black list dataset. We have increased benign URLs using faizan git repo At last, we have increased more number of phishing URLs using Phishtank dataset and PhishStorm dataset As we have told you that dataset is collected from different sources. So firstly, we have collected the URLs from different sources into a separate data frame and finally merge them to retain only URLs and their class type.

### Data Transformation

In the Transformation Process, we will convert our original dataset which is in excel format to CSV Format.

### Data Insertion into Database

* + - Database Creation and connection - Create a database with name passed. If the database is already created, open the connection to the database.
    - Table creation in the database.
    - Insertion of files in the table

### Export Data from Database

Data Export from Database - The data in a stored database is exported as a CSV file to be used for data Pre-processing and Model Training.

### Data Pre-processing

The classical machine learning tasks like Data Exploration, Data Cleaning, Feature Engineering, Feature Selection, Feature scaling, Data Balancing using sampling techniques etc.

### Model Building

After all the data pre-processing we will find the best model data. Each try, algorithms

will be passed with the best parameters derived from Grid-Search. We will calculate the Accuracy scores for models and select the model with the best score. Then the best model will be saved for the prediction purpose.

### Data from User

Here we will collect data of user such as age, limit-balance, educational status , marital status, sex, payment statuses, bill amounts, paid amount against each bills of last six months.

* 1. **Data preparing**

Here given data will be undergone all the pre-processing techniques (3.5) which we done on the early available dataset.

* 1. **Model calling for the data**

The saved model will be called for the prediction on the given data.

### Predicted data

On the given data the loaded model will perform prediction.

### Saving output at Database

The given data with the newly predicted data will be inserted to the pre-defined table available in the database for future usages.

# Unit Test Cases

|  |  |  |
| --- | --- | --- |
| Test Case Description | Pre-Requisite | Expected Result |
| **Verify whether the Application URL is accessible to the user** | 1. Application URL should be defined | Application URL should be accessible to the user |
| **Verify whether the Application loads**  **completely for the user when the URL**  **is accessed** | 1. Application URL is accessible 2. Application is deployed | The Application should load completely for the user when the  URL is accessed |
| **Verify whether user is able to see input**  **fields on logging in** | 1. Application is accessible 2. User is able to see input fields. | User should be able to see input  fields on logging in |
| **Verify whether user is able to edit all**  **input fields** | 1. Application is accessible 2. User is able to see input fields. 3. User is able to edit input fields. | User should be able to edit all input  fields |
| **Verify whether user gets Submit**  **button to submit the inputs** | 1. Application is accessible 2. User is able to see input fields. 3. User is able to edit input fields. 4. User is able to see submit button. | User should get Submit button to  submit the inputs |
| **Verify whether user is presented with**  **prediction results on clicking submit** | 1. Application is accessible 2. User is able to see input fields. 3. User is able to edit input fields. 4. User is able to see submit button. | User should be presented with Predicted with results on clicking submit |

THE END