📌Phishing Detection System — Complete Final Year Project Guide

---

🔗Stage 1 — Project Setup

Tools & Requirements:

- Python 3.10+
- VS Code / PyCharm
- Git & GitHub
- Chrome (for extension)

Python Libraries:

```
pip install pandas numpy scikit-learn beautifulsoup4 requests whois flask
```

Folder Structure:

```
phishing_detection_system/
  ├── data/
  ├── notebooks/
  ├── feature_extractor/
  ├── model/
  ├── api/
  ├── extension/
  ├── requirements.txt
  ├── README.md
```

🔗Stage 2 — Dataset Collection

Sources:

- Phishing: PhishTank, OpenPhish, Kaggle
- Legit: Alexa Top Sites, Tranco

Goal:

- Collect \~2,000–5,000 URLs
- Store in CSV:

```
URL, Label
http://malicious.com, phishing
http://google.com, legit
```

🔗Stage 3 — Feature Extraction Module

Key Features:

- URL Length
- "@" symbol presence
- "-" in domain
- IP address in URL
- HTTPS usage
- Number of subdomains
- Suspicious keywords (login, verify)
- Domain age (WHOIS)

Example Python Snippet:

```python
from urllib.parse import urlparse
import whois

def extract_features(url):
    features = {}
    features['url_length'] = len(url)
    features['has_at'] = '@' in url
    parsed = urlparse(url)
    features['has_https'] = parsed.scheme == 'https'
    features['has_ip'] = parsed.netloc.replace('.', '').isdigit()
    features['num_subdomains'] = len(parsed.netloc.split('.')) - 2
    features['has_dash'] = '-' in parsed.netloc

    try:
        domain_info = whois.whois(parsed.netloc)
        if domain_info.creation_date:
            age = (domain_info.expiration_date - domain_info.creation_date).days
            features['domain_age'] = age
        else:
            features['domain_age'] = 0
    except:
        features['domain_age'] = 0

    return features
```

🔗Stage 4 — Data Preprocessing

- Extract features for all URLs.
- Save to `features.csv`:

```
url_length, has_at, has_ip, has_https, num_subdomains, domain_age, label
45, 0, 1, 0, 3, 0, phishing
```

- Encode booleans, normalize if needed.

🔗Stage 5 — Model Building & Training

Steps:

1. Load features with pandas
2. Train/test split (80/20)
3. Train Logistic Regression, Random Forest, or XGBoost
4. Evaluate performance (accuracy, precision, recall)
5. Save model with joblib

Example:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import joblib

# Load data
df = pd.read_csv('data/features.csv')
X = df.drop('label', axis=1)
y = df['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(accuracy_score(y_test, y_pred))

joblib.dump(model, 'model/phishing_model.pkl')
```

🔗Stage 6 — Testing & Evaluation

- Use confusion matrix, precision, recall.
- Make EDA notebook for analysis.

🔗Stage 7 — Backend API (Flask)

Example API:

```
from flask import Flask, request, jsonify
import joblib
from feature_extractor import extract_features

app = Flask(__name__)
model = joblib.load('model/phishing_model.pkl')

@app.route('/predict', methods=['POST'])
def predict():
    url = request.json['url']
    features = extract_features(url)
    input_data = [[features[k] for k in sorted(features)]]
    prediction = model.predict(input_data)
    return jsonify({'prediction': prediction[0]})

if __name__ == '__main__':
    app.run(debug=True)
```

🔗Stage 8 — (Optional) Chrome Extension

- Use JavaScript to get current URL.
- Call Flask API.
- Show Safe/Phishing alert.

Extension Folder:

```
extension/
 ├── manifest.json
 ├── popup.html
 ├── popup.js
```

🔗Stage 9 — Deployment

- Deploy Flask API on Heroku/Render.
- Document project clearly.
- Add README.md with:
- Project summary
- Setup instructions
- Screenshots
- Future scope

👙Final Deliverables

- Working ML model & API
- Feature extractor module
- Dataset CSVs
- Optional Chrome extension
- Demo video or screenshots
- GitHub repo with clear README

Good Luck! 🚀