Smart Email Guardian: AI-Powered Spam & Phishing Detection Toolkit

This challenge is designed for 3rd-year Computer Science students specializing in cybersecurity, cloud computing, mobile application development, and web development. It focuses on using AI in a CPU environment, encouraging research, practical Python use, and cross-disciplinary problem solving without prior machine learning experience.

Challenge Brief

Build a lightweight AI-enhanced tool that identifies phishing or spam messages using pre-trained models or public APIs. Develop a basic frontend (mobile or web), a secure cloud backend, and expose core features through a Python CLI and API. Emphasize security, usability, and CPU-only inference.

Learning Goals

- Apply pretrained AI tools without training from scratch
- Integrate Python tools with cloud and frontend technologies
- Work with security best practices (input validation, token auth)
- Build and document a full-stack mini-application
- Use GitHub for version control, documentation, and submission

Project Requirements

1. AI Logic (Python, CPU-only)

Use a pretrained model (e.g., HuggingFace `distilbert`, OpenAI API, or scikit-learn pipeline).

The AI logic must: - Accept raw email text as input - Return classification (phishing/spam/legit), confidence score, and explanation - Run entirely on CPU

2. CLI Tool Build `email_guard.py`, a command-line interface tool that:

- Accepts input text
- Prints a JSON-style output with prediction and explanation

3. Frontend Interface (Choose One)

- Web App (React, Flask, Streamlit, or Django)
- Mobile App (Kivy, Flutter, or React Native)

Functionality: - Accept user input - Display results from the backend - Optionally, visualize scan history/stats

4. Backend & API Create a RESTful backend (Flask/FastAPI/Node/Firebase Functions) that:

- Accepts scan requests via `/scan`
- Stores results via `/history`
- Uses a free-tier cloud platform (Replit, Railway, Render, Firebase, etc.)
- Implements API key/token-based access control

5. Security Requirements

- Sanitize input and escape HTML/JS if applicable
- Use HTTPS or secure tunnel if hosting locally
- Token-based or key-based API access control
- Include a `security_notes.md` describing threats and mitigations

Deliverables

Submit a public GitHub repository with the following structure: email_guard/ ai/email_guard.py frontend/ backend/app.py docs/ README.md architecture.png security_notes.md tests/test_email_guard.py requirements.txt reflection.md

Evaluation Criteria

Area	Points
AI integration (CPU-safe)	20
Clear and working backend	15
Frontend (UI/UX)	15
Input validation / security	15
Research explanation	10
Code structure & docs	10
GitHub usage	10
Bonus (SDK, video, Docker)	5

Bonus Extensions (Optional)

- Export a reusable Python SDK
- Dockerize the full stack
- Add scan history UI
- Add Slack bot or Gmail inbox integration