Low-Level Design: Email Delivery Optimizer

Harsh

Overview

The Email Delivery Optimizer is a multi-interface Python-based tool that helps users diagnose, improve, and manage email deliverability. It supports:

- Domain authentication checks (SPF, DKIM, DMARC)
- Sending authenticated HTML emails
- Analyzing inbound headers
- Using Web UI, Desktop GUI, and CLI
- A reusable footer brand mark on every page

Core Modules

core.diagnostics

Purpose: DNS-based checks for email authentication records.

- `spf`$_c$*hecker.py–UsesDNSresolvertofetchSPFrecord*`dkim`$_c$*hecker.py–Buildsdomainandfetches*$DKIMre$

core.formatter

Purpose: Email template generation and sending.

- `email_template_formatter.py` – Formats travel HTML emails
- `send_email.py` – Sends email via Gmail SMTP with app password

core.imap

Purpose: Analyzes inbound Gmail headers via IMAP.

- `analyzer.py` – Extracts header authentication results

Web Interface (Flask)

Routes:

- `/` – Home + diagnostics form
- `/email` – Send branded emails
- `/analyze` – Analyze inbound headers
- `/setup` – Save session-based config

Templates

`index.html`, `results.html`, `email_preview.html`, `analyze_form.html`, etc.

GUI Interface (tkinter)

Standalone GUI app with domain checker + email composer, uses ‘core‘ logic directly.

CLI Interface

Run as:

```
python main.py --mode check --domain example.com
```

Style and Config

CSS classes: instructions, status-pass/fail, suggestion, footer, setup-btn, etc.
.env variables: `GMAIL_USER`, `GMAIL_APP_PASSWORD`

Dependencies

```
flask
rich
requests
dnspython
python-dotenv
imapclient
pyzmail36
```

Security Considerations

- No storage of credentials
- Gitignored .env
- Safe input handling

Future Enhancements

PDF reports, visual diagnostics, '.eml' parsing, AI-powered insights, domain rep score.

Design Choices

Flask + CLI + GUI hybrid model. Reusable modular logic. Plug-and-play: users configure Gmail and domain at runtime. Creator signature in all templates.