# **Low-Level Design: Email Delivery Optimizer**

## **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

## **Core Modules**

### **1. core.diagnostics**

Purpose: DNS-based checks for email authentication records.

* spf\_checker.py  
  + Function: get\_spf\_record(domain)
  + Uses dns.resolver to fetch TXT records
  + Parses and returns SPF record
  + Used by: Flask /, GUI, CLI check
* dkim\_checker.py  
  + Function: get\_dkim\_record(domain, selector)
  + Constructs selector.\_domainkey.domain, queries DNS
  + Returns DKIM record or error
  + Used by: Flask /, GUI, CLI check
* dmarc\_checker.py  
  + Function: get\_dmarc\_record(domain)
  + Checks \_dmarc.domain for TXT record
  + Parses policy and reporting URIs
  + Used by: Flask /, GUI, CLI check

### **2. core.formatter**

Purpose: Composing and sending structured emails

* email\_template\_formatter.py  
  + Function: generate\_email\_html(name, itinerary, price)
  + Outputs branded HTML template string
  + Used by: Flask /email, GUI, CLI send
* send\_email.py  
  + Function: send\_email(to, subject, html, from\_email, app\_password)
  + Uses smtplib.SMTP\_SSL with Gmail SMTP (port 465)
  + Authenticates using App Password
  + Used by: Flask /email, GUI, CLI send

### **3. core.imap**

Purpose: Inbound email header analysis

* analyzer.py  
  + Function: fetch\_latest\_email(email, password, subject\_filter)
  + Uses imapclient to access inbox
  + Parses headers using pyzmail36
  + Extracts SPF, DKIM, Authentication-Results
  + Used by: Flask /analyze, GUI, CLI analyze

## **Web Interface (app/)**

### **routes.py**

Uses Flask Blueprint with 3 primary routes:

* /  
  + Accepts domain & selector
  + Calls get\_spf\_record, get\_dkim\_record, get\_dmarc\_record
  + Renders results.html
* /email  
  + HTML form for composing travel email
  + Calls generate\_email\_html, send\_email
  + Renders preview
* /analyze  
  + Accepts Gmail + App Password
  + Calls fetch\_latest\_email
  + Renders analysis

### **templates/**

* index.html — Home form
* results.html — Diagnostic report
* email\_preview.html — Email HTML output
* analyze\_form.html — Gmail login
* analyze\_result.html — Header interpretation

## **GUI Interface (gui/)**

* Built using tkinter
* Input forms for:  
  + Domain checking (spf, dkim, dmarc)
  + Email send (HTML + Gmail)
  + Header analysis (inbound)
* Core logic reused from core/

## **CLI Interface (main.py)**

* Uses argparse
* Modes:  
  + check → diagnostics
  + send → send mail
  + analyze → header parser
* Directly imports from core/

## **Static & Config**

### **static/style.css**

* Shared CSS styles for results and pages

### **.env (Optional)**

* Environment variables for Gmail user + password
* Loaded with python-dotenv

## **Dependencies**

flask

rich

requests

dnspython

python-dotenv

imapclient

pyzmail36

## **Security Considerations**

* App Passwords used, not stored
* .env excluded via .gitignore
* Minimal access to only email headers

## **Summary of Class/Script Interactions**

| **Component** | **Calls** | **Dependencies** |
| --- | --- | --- |
| Web (routes.py) | diagnostics, formatter, imap | flask, jinja, core modules |
| GUI | diagnostics, formatter, imap | tkinter, core modules |
| CLI | diagnostics, formatter, imap | argparse, core modules |
| Core | DNS, SMTP, IMAP, pyzmail | dns.resolver, smtplib, imapclient |

## **Future Enhancements**

* PDF/HTML audit exports
* AI-based recommendations
* Header analysis from .eml
* Dashboard with auth result charts

## **Key Design Decisions**

* Use Flask for web deployment
* Shared logic across CLI, GUI, Web
* Simple IMAP+SMTP+DNS focus for portability
* No databases or login flows for simplicity