

# Technical Documentation — Bulk Email Campaign Management System

**Version: 1.0**

**Tech Stack: Django, Celery, Redis, SMTP, IMAP, PostgreSQL**

**Author: Pragatham Solution & Services OPC Pvt Ltd**

---

## 1. System Components Overview

Component	Description
<b>Django (Web App Layer)</b>	Handles UI, forms, dashboards, models, recipient uploads, campaign creation
<b>Celery Worker</b>	Executes scheduled tasks such as sending emails
<b>Celery Beat</b>	Periodic scheduler for triggering email campaigns and bounce processing
<b>Bounce Processor (IMAP Service)</b>	Connects to Gmail IMAP and extracts delivery failures
<b>SMTP Email Service</b>	Sends emails using Gmail SMTP with App Password
<b>PostgreSQL</b>	Stores campaigns, recipients, logs, bounce details
<b>MaterializeCSS UI</b>	Modern frontend for admin dashboards and forms

---

## 2. Modules and Technical Responsibilities

### 2.1 Campaign Service (`campaigns.services`)

Responsible for all business logic related to campaign creation, scheduling, logs, and report generation.

## Key Functions

Function	Purpose
<code>create_campaign()</code>	Validates and saves a new campaign
<code>attach_recipients_to_campaign()</code>	Assigns recipients/groups to campaign
<code>get_scheduled_campaigns()</code>	Fetches campaigns ready to send
<code>update_campaign_status()</code>	Sets status to Draft / Scheduled / In Progress / Completed
<code>generate_campaign_report()</code>	Returns summary data for email/report export

## Important Notes

- Non-UI logic resides here (Separation of Concerns)
  - Celery tasks call service functions to guarantee uniform behavior
- 

## 2.2 Recipient Service (`recipients.services`)

### Responsibilities

- CSV parsing
- Validation (email format, duplicates, subscription)
- Bulk creation of recipients
- Group management
- Mapping recipients to campaigns

## Key Functions

Function	Purpose
<code>parse_csv(file)</code>	Validates CSV file and extracts rows
<code>bulk_insert_recipients(rows)</code>	Inserts thousands of rows efficiently
<code>assign_group(recipient, group)</code>	Adds recipients to a group
<code>get_subscribed_recipients()</code>	Used by campaign senders

## Validation Logic

- Checks email format via regex
  - Prevents duplicate emails
  - Ensures subscription status is valid
- 

## 3. Data Models (ORM)

### 3.1 Campaign Model

```
class Campaign(models.Model):  
    name  
    subject  
    content  
    scheduled_time  
    status # Draft/Scheduled/In Progress/Completed  
    created_at
```

### 3.2 Recipient Model

```
class Recipient(models.Model):  
    name  
    email  
    subscription_status # subscribed/unsubscribed  
    groups = ManyToMany
```

### 3.3 CampaignRecipient Model

Tracks individual email send results.

```
class CampaignRecipient(models.Model):  
    campaign  
    recipient  
    status # sent/failed/pending  
    fail_reason  
    sent_at
```

### 3.4 BounceRecord Model

```
class BounceRecord(models.Model):  
    campaign  
    recipient_email  
    reason  
    processed_at
```

---

## 4. Django Forms

### 4.1 CampaignForm

Handles campaign creation/editing.

#### Features:

- Rich textarea for HTML email content
- DateTime picker
- Group / Recipient selector
- Early validation (missing fields, invalid schedule)

```
class CampaignForm(forms.ModelForm):  
    content = forms.CharField(widget=forms.Textarea)
```

---

### 4.2 RecipientUploadForm

Used in CSV Upload UI.

#### Validates:

- File extension
- File size
- Email duplication

---

### 4.3 AddRecipientsToCampaignForm

Dropdown to select campaign and attach recipients.

---

## 5. Services: Email Sending Pipeline

### 5.1 SMTP Email Service (`email_service.py`)

Handles:

- Gmail SMTP login
- Sending HTML emails
- Attaching campaign metadata in subject (CID tag)

**Example:**

Subject: Offer Zone [CID:18]

This enables IMAP to later identify campaign-related bounces.

**Core Function**

```
send_email(to, subject, html_body)
```

Includes retry logic for timeouts/SMTP errors.

---

## 🔥 6. Celery Task Architecture

### 6.1 Email Sending Task (`send_campaign_emails`)

Triggered when campaign is due.

**Flow:**

1. Get all `CampaignRecipient` entries where `status = pending`
2. For each recipient:
  - o Send email
  - o Update sent or failed status
3. Update campaign's `sent_count` and `failed_count`
4. Mark campaign as Completed

**Task Code Summary:**

```
@app.task
def send_campaign_emails(campaign_id):
    campaign = Campaign.objects.get(id=campaign_id)
    recipients = CampaignRecipient.objects.filter(campaign=campaign)
    for r in recipients:
        try:
            send_email(...)
            r.status = "sent"
        except Exception as e:
            r.status = "failed"
            r.fail_reason = str(e)
        r.save()
```

---

### 6.2 Scheduler Task (`run_scheduled_campaigns`)

**Runs every minute:**

- Finds campaigns where:

```
status = "Scheduled"  
scheduled_time <= now
```

- Starts Celery task `send_campaign_emails()`
- 

### 6.3 Bounce Processing Task (`process_bounces`)

Runs every 5 minutes

- Logs into Gmail IMAP
- Searches for subjects containing "**Delivery Status Notification (Failure)**"
- Extracts:
  - failed email
  - campaign ID from subject
  - bounce reason

Saves results:

```
BounceRecord.objects.create(  
    campaign=campaign,  
    email="user@example.com",  
    reason="Mailbox unavailable"  
)
```

---

## 7. IMAP Bounce Processing Service

Located in:

`campaigns/imap_bounce_processor.py`

Responsibilities

- Login IMAP
- Fetch unread bounce messages
- Parse MIME payload
- Extract:
  - Original recipient
  - SMTP status code
  - Human-readable failure description
  - CID (campaign ID)

Regex Used:

```
CID_PATTERN = r"\[CID:(\d+)\]"
```

---

## 8. Frontend Layer

Built using **MaterializeCSS** with custom UI enhancements.

Screens include:

- [Dashboard](#)
- [All Campaigns](#)
- [Create Campaign](#)
- [Upload Recipients](#)
- [Failed Reports](#)
- [Settings](#)

Frontend is intentionally lightweight:

- HTML Templates
  - Django Template Tags
  - No React or Vue to keep project simple
- 

## 9. Dashboard Metrics Calculation

Dashboard View aggregates data using Django ORM:

```
total_recipients = Recipient.objects.count()
total_campaigns = Campaign.objects.count()
sent_count = CampaignRecipient.objects.filter(status="sent").count()
failed_count =
    CampaignRecipient.objects.filter(status="failed").count()
```

Displayed in cards.

---

## 10. Reports

### 10.1 Bounce Report CSV

Generated from the BounceRecord table.

### 10.2 Campaign Summary Report

Includes:

- Total recipients
- Sent successfully
- Failed
- Percentage success
- Failure reasons

Automatically emailed to admin after campaign completion.

---

## 11. Security Considerations

- Gmail App Password (not normal password)
  - No direct email entry on front end without validation
  - Duplicate-prevention in recipients
  - Throttling in SMTP using Celery retry
  - Campaign content sanitized to avoid XSS
- 

## 12. Error Handling

### SMTP Errors:

- Timeout
- Invalid credentials
- Rate limits
- Gmail anti-spam rejections

Handled using:

```
try:  
    send_email()
```

```
except Exception as e:  
    log.failure  
    retry (limited)
```

#### IMAP Errors:

- Connection blocked
- Parsing failure
- Empty results

Logged in Celery and stored in database.

---

## 13. Performance Considerations

- Batch inserts for recipients
  - Parallel Celery workers
  - Minimized DB queries using `select_related()`
  - Recipient filtering using indexed fields
  - IMAP processed only unread messages for efficiency
- 

## 14. Recommended Folder Structure

Megovation

```
bulk_email_system  
    __init__.py  
    asgi.py  
    celery.py  
    requirements.txt  
    settings.py  
    urls.py  
    wsgi.py  
campaigns  
management  
migrations
```

```
static
templates
__init__.py
admin.py
apps.py
context_processors.py
forms.py
imap_bounce_processor.py
models.py
readme.py
README_DOCS.md
services.py
tasks.py
tests.py
urls.py
views.py
email-dev-task
static
venv
celerybeat-schedule.bak
celerybeat-schedule.dat
celerybeat-schedule.dir
db.sqlite3
manage.py
```

---

## ⌚ 15. Conclusion

This email campaign engine is:

- Fully automated
- Scalable
- Data-driven
- Enterprise-grade
- Cleanly separated into services, tasks, forms, UI layers

An ideal structure for real-world mass-mailing, job portals, ed-tech notifications, or system alerts.

For Production we can use pgsql