ChatGPT

# Phase 12.2: Bulk Email Campaign System

This design adds a full-featured bulk email campaign module to Arcana, allowing creative teams to compose, preview, and send mass emails to contacts (with dynamic merge fields), plus monitor delivery, opens, and failures. It includes template management, campaign creation/launch, background sending with throttling, and compliance (unsubscribe links). The system uses three new tables (`email_templates`, `email_campaigns`, `email_recipients`) linked to Arcana's existing contacts/events.

## Database Schema

```
-- Store reusable email templates (with both HTML and plain-text)
CREATE TABLE email_templates (
  id             SERIAL PRIMARY KEY,
  name           VARCHAR(255) NOT NULL,
  subject        VARCHAR(255) NOT NULL,
  body_html      TEXT NOT NULL,
  body_text      TEXT NOT NULL,
  created_by     INTEGER NOT NULL REFERENCES users(id),
  created_at     TIMESTAMP NOT NULL DEFAULT NOW(),
  updated_at     TIMESTAMP NOT NULL DEFAULT NOW()
);

-- Email campaigns that use a template to send to a segment of contacts
CREATE TABLE email_campaigns (
  id             SERIAL PRIMARY KEY,
  template_id    INTEGER NOT NULL REFERENCES email_templates(id),
  created_by     INTEGER NOT NULL REFERENCES users(id),
  subject_override VARCHAR(255),          -- optional override for subject
  target_filter  JSONB,                   -- e.g. {"tags": ["client"],
"event_id": 123}
  status         VARCHAR(20) NOT NULL,    -- e.g. 'draft', 'scheduled',
'sending', 'completed', 'cancelled'
  created_at     TIMESTAMP NOT NULL DEFAULT NOW(),
  updated_at     TIMESTAMP NOT NULL DEFAULT NOW()
);

-- Recipients of each campaign (one row per contact per campaign)
CREATE TABLE email_recipients (
  id             SERIAL PRIMARY KEY,
  campaign_id    INTEGER NOT NULL REFERENCES email_campaigns(id),
  contact_id     INTEGER NOT NULL REFERENCES contacts(id),
```

```
  delivery_status VARCHAR(20) NOT NULL,
-- e.g. 'pending', 'sent', 'delivered', 'opened', 'bounced', 'failed'
  error_message  TEXT,                    -- store SMTP or API error if any
  opened_at      TIMESTAMP,               -- timestamp of first open via
tracking pixel
  created_at     TIMESTAMP NOT NULL DEFAULT NOW()
);
CREATE INDEX ON email_recipients(campaign_id);
CREATE INDEX ON email_recipients(contact_id);

-- (Optional) Unsubscribe tracking (or could add a flag to contacts table)
CREATE TABLE email_unsubscribes (
  id           SERIAL PRIMARY KEY,
  contact_id   INTEGER NOT NULL REFERENCES contacts(id),
  unsubscribed_at TIMESTAMP NOT NULL DEFAULT NOW()
);
```

- **Templates**: `body_html` and `body_text` support Handlebars-style merge tags like `{{contact.first_name}}` or `{{event.date}}`. At send time, each tag is replaced with the recipient's data [1].
- **Campaigns**: The `target_filter` JSON encodes which contacts to include (e.g. by event ID, tags, or any query on Arcana's contact/event fields).
- **Recipients**: Each campaign's recipients are pre-populated when the campaign is launched. `delivery_status` tracks progress. For example, initially `'pending'`, then `'sent'` once the worker sends the email, and `'opened'` when an open is detected. `error_message` logs failures (bounces, etc.).

All timestamps (`created_at`, `updated_at`, etc.) allow auditing. Foreign keys link to Arcana's existing `contacts(id)`, `events(id)`, and `users(id)` tables. This cleanly ties templates and campaigns to Arcana data (e.g. fetching `event.date` for each contact's event).

## API Endpoints

**Email Templates API:**
- `GET /api/email_templates` – List saved templates.
- `GET /api/email_templates/{id}` – Get one template.
- `POST /api/email_templates` – Create a template. *Body:* `{ name, subject, body_html, body_text }`.
- `PUT /api/email_templates/{id}` – Update a template (allow editing name, subject, body).
- `DELETE /api/email_templates/{id}` – (Optional) Delete a template.

**Campaigns API:**
- `GET /api/email_campaigns` – List campaigns (with status and counts).
- `GET /api/email_campaigns/{id}` – Get campaign details and stats (including total/failed/open counts).
- `POST /api/email_campaigns` – Create a campaign. *Body:* `{ template_id, subject_override?,`

`target_filter (JSON) }`. Status defaults to `"draft"`.
- `PUT /api/email_campaigns/{id}` – Update (e.g. edit subject_override or filter while in draft).
- `POST /api/email_campaigns/{id}/launch` – Launch (send) a campaign: sets `status='sending'`, queries contacts matching `target_filter`, creates `email_recipients` rows, and enqueues sending jobs.
- `POST /api/email_campaigns/{id}/test_send` – Send test email(s) using this template (body/subject) to the current user or a provided email list. *Body:* `{ test_emails: ["me@example.com"] }`. Useful for previewing.

**Recipients/Tracking API:**
- `GET /api/email_campaigns/{id}/recipients` – List recipients and their `delivery_status`. Useful for campaign dashboard (e.g. "23/30 delivered").
- **Open Tracking:** `GET /api/email/open?campaignId={id}&recipientId={id}` – Returns a 1×1 transparent pixel image. This endpoint updates that recipient's `opened_at` timestamp (if not already set). The image URL is embedded in the email body to track opens.
- **Unsubscribe:** `GET /api/email/unsubscribe?contactId={id}` – (Can also include campaignId as query). When a user clicks the unsubscribe link, this marks the contact as unsubscribed (e.g. insert into `email_unsubscribes`). Future campaigns should filter these contacts out. *Under CAN-SPAM, all marketing emails must contain an easy unsubscribe link* [2]. The UI will generate links like:

```
<a href="https://app.arcana.com/api/email/unsubscribe?
contactId={{contact.id}}">Unsubscribe</a>
```

These endpoints fit RESTful conventions (similar to Mailchimp's API) and integrate Arcana's auth and data (contacts, events). For example, `target_filter` JSON might be `{ "tags": ["client"], "event_id": 123 }`, and the backend would query `contacts` where `contact.event_id = 123 AND 'client' = ANY(contact.tags)`.

## React Components (UI)

The frontend will include these main components (using React or similar):

- `EmailTemplateEditor` – A form for creating/editing templates. It includes fields for Template Name, Subject, and a rich-text HTML editor plus plain-text view. It supports inserting merge tags via a dropdown (e.g. a button "Insert merge field" that lists `client.name`, `event.date`, etc). It also shows a live preview. *Scaffold example:*

```
function EmailTemplateEditor({ template, onSave }) {
  const [name, setName]       = useState(template?.name || '');
  const [subject, setSubject] = useState(template?.subject || '');
  const [bodyHtml, setBodyHtml] = useState(template?.body_html || '');
  // Render a rich text editor (e.g. using react-quill or DraftJS)
  return (
    <div>
```

```
      <input value={name} onChange={e => setName(e.target.value)}
placeholder="Template Name" />
      <input value={subject} onChange={e => setSubject(e.target.value)}
placeholder="Email Subject" />
      <RichTextEditor value={bodyHtml} onChange={setBodyHtml}
insertMergeField={(field)=>{...}} />
      <button onClick={() => onSave({ name, subject, body_html: bodyHtml,
body_text: plainText(bodyHtml) })}>
        Save Template
      </button>
    </div>
  );
}
```

The `RichTextEditor` shows a split view: HTML editing pane on left and real-time rendered preview on right.

- `EmailCampaignForm` – A page/modal for creating a new campaign. It lets the user: choose a saved template (dropdown), override subject if desired, define the target segment (via a filter builder UI, e.g. pick event or tags from Arcana data), and preview a sample email (filling merge fields with placeholder text). Also buttons for "Send Test Email" and "Launch Campaign". Example:

```
function EmailCampaignForm({ templates, contacts, events, onLaunch }) {
  const [templateId, setTemplateId] = useState();
  const [subject, setSubject] = useState('');
  const [filter, setFilter] = useState({}); // e.g. {event_id:123, tags:
['client']}
  return (
    <div>
      <select onChange={e => setTemplateId(e.target.value)}>...</select>
      <input placeholder="Subject override" value={subject} onChange={...}/
>
      <SegmentBuilder filter={filter} onChange={setFilter} />  {/* e.g. UI
to pick tags, events */}
      <button onClick={()=>sendTest(templateId, subject)}>Send Test</
button>
      <button onClick={()=>onLaunch({ template_id: templateId,
subject_override: subject, target_filter: filter })}>
        Launch Campaign
      </button>
    </div>
  );
}
```

- `CampaignDashboard` – Shows live status of a campaign. It displays counts like "Delivered: 23, Opened: 5, Bounced: 2, Pending: 0". It refreshes periodically (polling `/email_campaigns/{id}` or using web sockets). It might show a table of recipients with their status. *Scaffold:*

```
function CampaignDashboard({ campaignId }) {
  const [stats, setStats] = useState({});
  useEffect(() => {
    fetch(`/api/email_campaigns/${campaignId}
`).then(res=>res.json()).then(setStats);
  }, [campaignId]);
  return (
    <div>
      <h2>Campaign Status: {stats.status}</h2>
      <div>Sent: {stats.sentCount} / {stats.totalRecipients} delivered</
div>
      <div>Opened: {stats.openedCount}</div>
      <div>Errors: {stats.failedCount}</div>
      {/* Could also show a progress bar or chart */}
    </div>
  );
}
```

- `RichTextEditor` – A custom or third-party rich-text editor component that supports merging. It should allow users to insert placeholders (merge tags) at the cursor. Under the hood, we use a library (e.g. React Quill or Draft.js) and extend it to insert tokens like `{{contact.first_name}}`. The saved HTML should preserve these `{{ }}` tokens exactly.

Other components/pages include: a list of Campaigns (`EmailCampaignList`), a modal for test-send input, and an Unsubscribe landing page (or confirmation) which calls the unsubscribe API and thanks the user.

Altogether, the UI flows: design/edit templates, create campaigns (with preview and test-send), then a dashboard for each live campaign. The WYSIWYG editor ensures creatives don't have to write raw HTML, but any valid HTML with placeholders is allowed.

## Background Worker & Delivery Logic

Sending bulk email is handled asynchronously in a worker to avoid blocking the request/response cycle and to throttle volume. The launch endpoint enqueues a job (e.g. via a task queue like Celery/RabbitMQ or a Node job queue like Bull/Redis). The worker performs roughly:

1. **Initialize recipients**: Query contacts matching `target_filter`. For each, insert a row in `email_recipients (campaign_id, contact_id, delivery_status='pending')`. Also mark campaign `status = 'sending'`.

2. **Send loop (throttled)**: In batches or one-by-one (depending on load), load a set of pending recipients:

```
// Pseudocode for a Node worker
async function processCampaign(campaignId) {
  let recipients = db.query(`SELECT * FROM email_recipients WHERE
campaign_id=? AND delivery_status='pending'`, [campaignId]);
  for (let rec of recipients) {
    // Fetch contact and, if needed, related event data
    const contact = await db.contacts.find(rec.contact_id);
    const event = await db.events.find(campaign.target_filter.event_id);

    // Render template: replace merge tags with contact/event data
    const emailHtml = renderTemplate(campaign.template.body_html, {
contact, event });
    const emailText = renderTemplate(campaign.template.body_text, {
contact, event });
    const subject = campaign.subject_override || campaign.template.subject;

    try {
      // Send via chosen provider
      if (config.emailProvider === 'sendgrid') {
        await sendgrid.send({to:contact.email, subject, html:emailHtml,
text:emailText, from:config.fromAddress});
      } else {
        await nodemailer.send({to:contact.email, subject, html:emailHtml,
text:emailText});
      }
      // Mark as sent (will count as delivered)
      await db.query(`UPDATE email_recipients SET delivery_status='sent'
WHERE id=?`, [rec.id]);
    } catch (err) {
      // On error (e.g. SMTP bounce, SendGrid error), log and mark
      await db.query(
        `UPDATE email_recipients SET delivery_status='failed',
error_message=? WHERE id=?`,
        [err.message, rec.id]
      );
    }
    // Throttle: pause briefly to avoid hitting rate limits
    await sleep(config.emailThrottleMs);
  }
  // After all, update campaign status
  await db.query(`UPDATE email_campaigns SET status='completed' WHERE id=?
`, [campaignId]);
}
```

Here, we use a simple loop with a delay (`sleep`) to throttle. More advanced: use multiple worker threads or a job queue to parallelize with a fixed concurrency. **Best practice** is to limit send rate to avoid ISP throttling [3] and use exponential backoff on retries [4]. For example, if `send()` fails due to a transient error, retry a few times with increasing delays. Using a robust queue (e.g. RabbitMQ or Redis) ensures tasks aren't lost and can be retried if the worker crashes [5].

3. **Retries**: If an email fails due to a soft bounce (e.g. temporary SMTP error), schedule a retry (exponential backoff). Log permanent bounces as failures (so repeated attempts aren't made).

4. **Webhooks/Feedback**: (Optional) If using SendGrid or Mailgun, set up webhooks to receive bounce and open events, and update `email_recipients.delivery_status` accordingly. At minimum, embed a tracking pixel: the `/api/email/open` endpoint above marks `opened_at`.

**Configuration:** An environment variable (e.g. `EMAIL_PROVIDER`) switches between SMTP (using NodeMailer) or SendGrid API. *SendGrid is recommended for high volume, as it offers scalability and deliverability*, whereas Gmail/SMTP has strict daily limits [6]. For example:

```
const sendWithSendGrid = async (msg) => {
  sendgrid.setApiKey(process.env.SENDGRID_API_KEY);
  await sendgrid.send(msg);
};
const sendWithSMTP = async (msg) => {
  // nodemailer using SMTP config from env (host, user, pass)
  let transporter = nodemailer.createTransport({ /*...*/ });
  await transporter.sendMail(msg);
};
const sendEmail = (msg) => process.env.EMAIL_PROVIDER === 'sendgrid'
    ? sendWithSendGrid(msg)
    : sendWithSMTP(msg);
```

After sending each batch, the worker pauses or yields. According to deliverability research, **staggering batches** reduces spam flags [3].

If the worker detects too many failures in a row (e.g. blocked by an ISP), it should slow down or pause and alert an admin. Periodically, a cleanup job can purge old `email_recipients` or export campaign stats.

## Integration with Arcana Data

- **Merge Fields:** We expose Arcana contact fields (`first_name`, `last_name`, `email`, `company`) and event fields (`name`, `date`, `location`, etc.) as merge-tag variables. For example, `{{contact.first_name}}` or `{{event.date}}`. In the UI we might label them as "Client Name" and "Event Date." At send time, our `renderTemplate` function looks up these from the `contacts` and `events` tables.
- **Segmentation (`target_filter`):** The filter JSON may include Arcana-specific criteria (e.g. event date range, contact tag, project stage). The backend translates this into a SQL query on `contacts`

(joining `events` if needed). For example, selecting all contacts where `contact.status = 'active' AND event.date > today`. This way, campaigns seamlessly target the right Arcana contacts.

In sum, the campaign system uses Arcana's existing models as its data source, but adds new tables and logic for email-specific features.

## Sample Email Templates

Below are example templates relevant to creative operations. Each uses placeholders (`{{…}}`) for dynamic data:

- **Approval Reminder:**
  *Subject:* "Reminder: Approve Assets for **{{event.name}}**"
  *Body (HTML snippet):*

```
<p>Hi {{contact.first_name}},</p>
<p>This is a friendly reminder to review and approve the designs for
<strong>{{event.name}}</strong> scheduled on {{event.date}}. Please <a
href="{{approval_link}}">click here</a> to view the assets. Let us know if
you have any feedback.</p>
<p>Thank you!<br/>Creative Team</p>
<p><small><a href="https://app.arcana.com/api/email/unsubscribe?
contactId={{contact.id}}">Unsubscribe</a></small></p>
```

  *(The `{{approval_link}}` could be a special merge tag linking to the Arcana project's approval page.)*

- **Asset Handoff Notification:**
  *Subject:* "Final Assets Ready for **{{event.name}}**"
  *Body:*

```
<p>Hello {{contact.first_name}},</p>
<p>We've completed the final design assets for your event
<em>{{event.name}}</em> on {{event.date}}. You can download them here: <a
href="{{asset_download_url}}">Download Assets</a>.</p>
<p>If you have any last-minute changes, let us know. Otherwise, consider
this project finalized. Thank you for working with us!</p>
<p>Best regards,<br/>The Arcana Creative Team</p>
<p><small><a href="https://app.arcana.com/api/email/unsubscribe?
contactId={{contact.id}}">Unsubscribe</a></small></p>
```

- **Project Kickoff Announcement:**
  *Subject:* "Kickoff: **{{event.name}}** on {{event.date}}"
  *Body:*

```
<p>Dear {{contact.first_name}},</p>
<p>We're excited to begin work on <strong>{{event.name}}</strong>! Our
kick-off meeting is scheduled for {{event.date}} at {{event.location}}.
Please confirm your attendance.</p>
<p>Looking forward to a great collaboration.</p>
<p>Cheers,<br/>Your Project Manager</p>
<p><small><a href="https://app.arcana.com/api/email/unsubscribe?
contactId={{contact.id}}">Unsubscribe</a></small></p>
```

Each template includes an unsubscribe link in the footer (required by CAN-SPAM [2]). They show how merge fields like `{{contact.first_name}}` and `{{event.name}}` personalize the content.

This completes the design for Phase 12.2's bulk email campaign system, integrating with Arcana's existing data. It provides template creation, campaign launching, and robust sending/tracking, all following best practices (merge tags [1], unsubscribe compliance [2], background processing [5]).

**Sources:** Best practices and examples were adapted from email marketing and deliverability guidelines [2] [5], and merge-tag templating (Handlebars) documentation [1]. These informed the architecture and feature list.

---


[1] Handlebars Overview – Iterable Support Center
https://support.iterable.com/hc/en-us/articles/35601631606036-Handlebars-Overview

[2] When Is It OK To Send An Email Without An Unsubscribe Link? | Customer.io
https://customer.io/learn/deliverability/unsubscribe-link

[3] [4] [5] Improving Email Delivery with Celery for High Volume Campaigns | MoldStud
https://moldstud.com/articles/p-maximize-email-deliverability-how-to-use-celery-for-high-volume-campaigns-a-case-study

[6] Sending bulk emails in Node.js using SendGrid API - GeeksforGeeks
https://www.geeksforgeeks.org/node-js/sending-bulk-emails-in-node-js-using-sendgrid-api/