

A Website and Membership Engine for Classic Liberals Australia

The full website build — information architecture, design system and a forty-plus record policy library — together with a purpose-built WordPress plugin handling applications, approvals, seven branded email templates and an automatic newsletter.

40+

POLICY
RECORDS

4

DATABASE
TABLES

7

EMAIL
TEMPLATES

6

ADMIN
SCREENS

CLIENT

Classic Liberals Australia

LIVE AT

classicliberals.org.au

SCOPE

Full site + custom plugin

BUILT BY

Rishfotech Solutions

AT A GLANCE

A public platform, and the operations system behind it.

Classic Liberals Australia is a newly formed Australian political party. They needed a public website capable of carrying a large and growing policy library, and a way to take membership applications from that website, review each one, correspond with applicants at every stage, and reach their whole list when something new was published.

Rishfotech delivered both: the complete website, and a custom WordPress plugin built specifically for the membership workflow. The two were designed together — the plugin's forms are shortcodes that drop into the site's own layouts, and its email templates use the site's own navy and gold identity.

The website

Information architecture	Six-section structure with a hierarchical state framework built to expand nationally
Policy library	Forty-plus policies as structured records with individual pages and a paginated archive
Design system	Navy and gold identity applied consistently across site and correspondence
Editorial	Blog with author attribution, feeding the automatic newsletter
Responsive	Off-canvas navigation and adaptive layouts across breakpoints

The plugin

Applications	Nine-field form with state selection and mandatory disclosure consent
Approval workflow	Pending, active or rejected — with the correct branded email on each transition
Subscribers	Separate two-field capture, deployable anywhere by shortcode
Email engine	Seven database-stored templates with merge tokens, editable by the client
Broadcast	Group sending filtered by state, deduplicated, fully logged
Newsletter	Publishing an article emails the entire list automatically

WHAT THIS DOCUMENT COVERS

Both halves of the delivery: the website build in Sections 02 and 03, then the plugin from Section 04 onward — why it was built rather than bought, how it is architected, how the client manages it, and what it solves. This is a technical document about software and a website; it takes no position on the client's political subject matter.

INSIDE THIS DOCUMENT

Contents

01	The Brief	04
	A new party, a growing policy library, and a members list to build	
02	The Website — Information Architecture	05
	Six sections, a policy library, a state framework built to expand	
03	The Website — Design & Build	06
	Identity, navigation, responsive behaviour, editorial	
04	Why We Built a Custom Plugin	07
	What the off-the-shelf options could not do	
05	Plugin Architecture	08
	Module boundaries and separation of concerns	
06	The Data Model	09
	Four purpose-built tables and why they exist	
07	Public Capture	10
	Two forms, one submission pipeline	
08	The Caching Problem	11
	The bug that only broke for real visitors	
09	Replacing What the Nonce Protected	13
	Honeypot, timing check, per-IP throttle	
10	The Member Lifecycle	14
	Three states, and an email at every edge	
11	The Email Template Engine	15
	Seven templates, token merging, client-editable	
12	Broadcast & Automatic Newsletter	17
	Group sending and publish-triggered dispatch	
13	How the Client Manages It	18
	The day-to-day runbook, screen by screen	
14	Resilience & Self-Healing	20
	Migrations, table checks, repair, graceful failure	
15	Challenges We Faced — How We Solved Them	21
	Four real problems, worked end to end	
16	What This Solves	23
	The problems that no longer exist	

A NOTE ON EVIDENCE

Website statements describe the live production site. Plugin statements are drawn from the source at version 4.2.1. No membership numbers, traffic figures or delivery rates are claimed, because none have been measured for this project.

SECTION 01

A new organisation, building its public presence from zero.

Two problems arrived together: publish a large and growing body of policy in a way people can actually navigate, and turn readers into a reviewed, contactable membership.

A newly formed political party has a specific and unusual content problem. Its entire proposition is its policy positions — not a product catalogue, not a service list, but dozens of individual position papers that need to be readable one at a time and browsable as a body of work. And that library grows continuously as the organisation develops.

The second problem is membership. Unlike a commercial signup, party membership is **applied for**. Somebody reads each application and decides. That single fact shaped everything on the operational side of the build.

The three requirements

01**A publishing platform**

Policy content that can grow indefinitely without the navigation degrading or anyone rebuilding a page.

02**A reviewed membership**

Applications that arrive pending, get reviewed by a human, and generate the right correspondence whichever way the decision goes.

03**A communication channel**

The ability to reach the entire list — deliberately when needed, and automatically when something is published.

The constraint that mattered most

This is a small organisation without in-house technical staff. Everything delivered had to be operable by a non-technical administrator: adding a policy, publishing an article, reviewing an application, editing the wording of a confirmation email, emailing everyone in a particular state. If any of those required a developer, the platform would have failed within a month of handover.

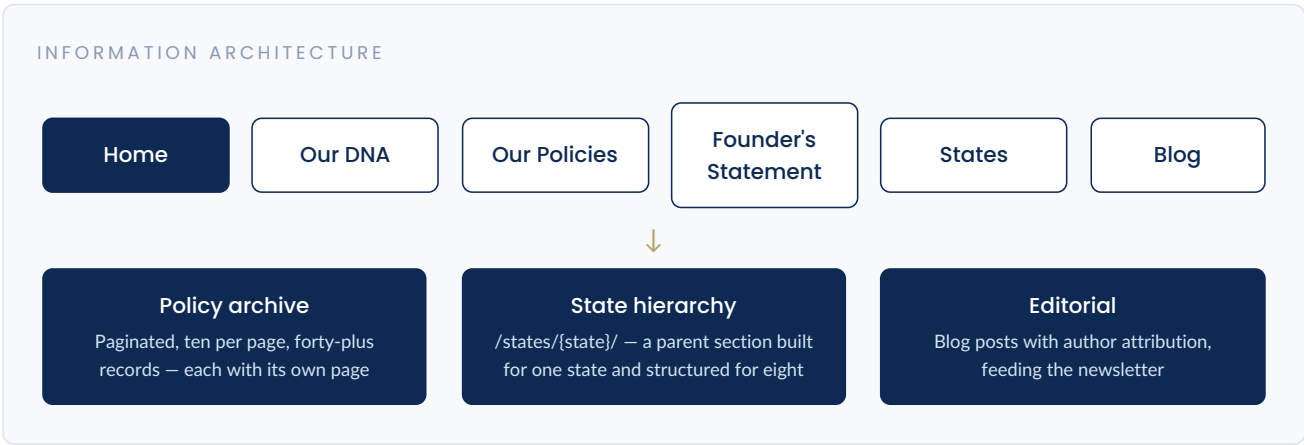
THE SHAPE OF THE ANSWER

Build the website so that repeating content — policies, articles, states — is **structured records rather than hand-built pages**. Then build the membership system as a dedicated plugin owning its own data, rather than bending a general-purpose membership product into a shape it was not designed for.

SECTION 02

Six sections, and a library that can keep growing.

The navigation is deliberately short. Everything that grows sits behind a structured archive rather than being added to the menu.



The policy library

Policies are the largest and fastest-growing body of content on the site, so they are modelled as structured records rather than as pages. Each policy has its own URL beneath the archive — `/our-policies/{policy}/` — and the archive itself paginates at ten per page, currently running to five pages.

Adding a policy is a content task	Creating one record publishes a full page, inserts it into the archive in the correct position, and updates pagination. No template work, no menu edit, no layout change.
Each policy is independently linkable	A single position can be shared, cited or linked to directly — which matters enormously for an organisation whose whole activity is having its positions read and debated.
The menu never grows	Forty policies do not become forty menu items. The navigation stays at six entries no matter how large the library becomes.

Built for national expansion

The States section currently carries a single state. It is structured as a parent section with state pages beneath it, not as a standalone page — so adding the remaining states as the party expands is a content operation against an architecture that already anticipates them. This was a deliberate decision at build time: the cost of structuring it correctly on day one was almost nothing, and the cost of restructuring it later, with live URLs in circulation, would have been considerable.

SECTION 03

An identity that had to work in two places at once.

Navy and gold, applied to the website and then carried through into every email the membership system sends – so correspondence reads as coming from the organisation, not from a plugin.

Deep navy

Headers, footers, primary panels and every email header. The dominant surface colour across both site and correspondence.

Gold

Accent rules, badges, numbered markers and the top bar on every email template. Used sparingly, which is what keeps it meaningful.

Light neutral

Page and email backgrounds, table striping and card fills – a soft blue-grey rather than pure white, so navy panels sit on it comfortably.

Navigation and responsive behaviour

Short primary menu	Six items plus a persistent <i>Join Us Now</i> action. The membership call to action is present in the header on every page and repeated in the footer.
Off-canvas mobile navigation	The menu collapses to a slide-in panel rather than a stacked dropdown, so a long list never pushes page content off the screen.
Structured footer	Three columns – the party sections, the states covered, and contact – with the membership action repeated beneath them.
Custom breakpoints	The build uses additional breakpoints beyond the defaults, so the policy archive and the long-form policy pages both reflow properly on tablet widths rather than only at phone and desktop.

Content-first page design

The homepage and the long-form pages – Our DNA, the Founder's Statement, individual policies – are set for sustained reading. This is a site whose visitors arrive to read arguments, not to scan a product page, and the typographic treatment reflects that: generous measure, clear heading hierarchy, and no competing interface elements alongside the text.

WHERE THE TWO HALVES MEET

The membership and subscriber forms are exposed as shortcodes, which means they drop into any page layout the site uses without special handling. The site's identity and the plugin's email templates share the same palette by design – an applicant who submits a form on the website receives a confirmation that looks like it came from the same organisation.

NOTE ON EVIDENCE

This document contains no website screenshots. Descriptions of layout and behaviour are drawn from the live production site.

SECTION 04

Why we built it rather than bought it.

Custom development is the more expensive answer and should need justifying. Here is the justification.

The plugin ecosystem has a great many membership plugins, almost all built around one of two models: instant self-service registration, or paid subscription that activates on payment. Party membership is neither. It is **applied for and granted** – and that difference is not a setting you can toggle.

What we evaluated, and why each was rejected

OPTION	WHY IT DID NOT FIT
A membership plugin	Built around user accounts, logins and access control. CLA members never log in and there is nothing gated to protect. It would have imported an entire authentication surface to solve a problem that is really a records-and-correspondence problem.
A form plugin	Captures a submission and emails it. No concept of a record with a reviewable status, no approve or reject action, no way to email everyone who has been approved.
An email marketing service	Good at broadcasting, but the list lives with a third party. It has no view of application status, so approvals could not drive segmentation, and the organisation's membership register would sit outside its own control.
Several of the above, combined	The most common approach and the worst outcome: three products to license, three to keep updated, data split across all of them, and a workflow held together by manual copying between systems.

The four things only a custom build gave us

01

The right data model

A member is a person with a postal address and a consent record – not a username and password. The schema matches the organisation, not a vendor's idea of one.

02

Status-driven correspondence

Approval and rejection each fire their own branded template automatically – the core of the workflow, and what no combination of stock plugins provides.

03

Data ownership

The register lives in the organisation's own database – no third-party terms, and no subscription that can lapse and take the list with it.

04

No upgrade roulette

Nothing in the workflow depends on a third party's roadmap. No vendor can deprecate a feature the organisation relies on.

THE HONEST TRADE-OFF

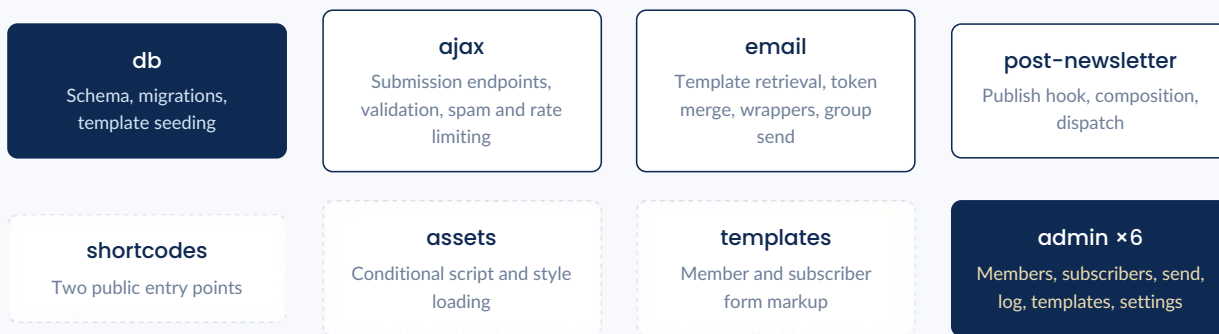
A custom plugin means Rishfotech maintains it. That is why it was built to report its own state, migrate its own schema and repair its own tables from a button – so routine operation never requires us.

SECTION 05

Module boundaries.

The plugin is split by responsibility rather than by feature, so that a change to how email renders never touches how a form is submitted.

MODULE MAP



Two decisions worth explaining

Admin assets load on six screens only

The admin stylesheet and script are enqueued against an explicit allow-list of plugin page hooks. Nothing this plugin owns is loaded anywhere else in the WordPress admin — no global bloat, and no chance of a style collision with another plugin's screens.

Templates live in the database, not in files

Storing the seven email templates as rows rather than PHP files is what makes them editable by the client. A template is addressed by a stable slug, so the code that sends the approval email refers to `member_approved` and cannot break if rows are reordered or reseeded.

DEVELOPMENT CONTEXT

Every entry point begins with an `ABSPATH` guard, so no file in the plugin can be executed by requesting it directly. Every admin action is gated on `manage_options` in addition to its nonce check — capability and intent are verified separately rather than assuming one implies the other.

SECTION 06

Four purpose-built tables.

None of this is stored as WordPress users or post meta. Members are not site users — they never log in — and modelling them as such would have imported a large amount of behaviour the organisation does not want.

SCHEMA

cla_members

Identity, postal address,
consent, status, subscription
fields

cla_subscribers

Name and email

cla_email_templates

Slug, label, subject, body

cla_email_log

Group, subject, body, sent
and fail counts, sender,
timestamp

DESIGN CHOICE

REASONING

Unique index on email

On both members and subscribers. A repeat submission updates the existing row rather than creating a duplicate — so somebody who applies twice does not appear twice in an export or receive two copies of a broadcast.

Status as a string, not a boolean

pending, active and rejected are three distinct states, and rejected is not the same as absent. Keeping rejected applications on file with their own status preserves the record.

The log stores the body

Not just the subject and a count. Six months later, the client can see exactly what was sent, to which group, by whom — which matters for an organisation that keeps records of its communications.

Subscription fields present but dormant

The members table carries subscription status, amount and next-payment date, and the settings screen carries a master switch that is off by default. The schema was built to accommodate recurring contributions without a migration when the organisation is ready for them.

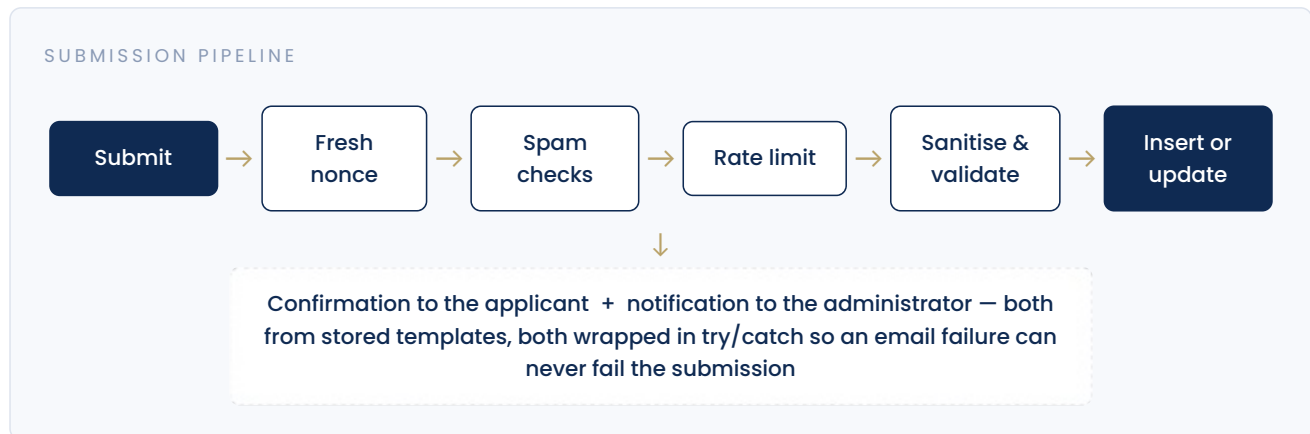
DEDUPLICATION IS A DATA-MODEL DECISION, NOT A FEATURE

A person can be both a member and a subscriber. Every send path in the plugin — group email and automatic newsletter alike — collects recipients into a map keyed on the lower-cased, trimmed email address before sending. Nobody receives the same message twice, and the behaviour is guaranteed by the structure rather than by remembering to check.

SECTION 07

Two forms, one pipeline.

The membership form and the subscriber form look nothing alike and collect very different data, but they travel through an identical submission path.



The membership form

Nine fields: title, first name, family name, email, street address, city or suburb, state, postcode, and a consent checkbox. Seven are required, and the consent checkbox is one of them – the submission is rejected outright without it, with an explicit message rather than a silent failure.

State is a controlled dropdown of the eight Australian states and territories. That is not a presentational choice: broadcast email can be filtered by state, and free-text entry would have produced "Vic", "VIC" and "Victoria" as three separate segments within a week.

Both forms are shortcodes

Each form is exposed as a shortcode, so the client can place either one on any page, in any layout, without a developer. Both submit over AJAX, so neither triggers a page reload, and both render server responses inline – validation errors, success messages and network failures all appear in the same place beneath the form.

ERRORS THAT SAY WHAT WENT WRONG

The front end distinguishes between HTTP status codes rather than showing one generic failure. A 403 reports that a security plugin may be blocking the request; a 404 reports that `admin-ajax.php` may be blocked; a 500 points at the PHP error log. On a site whose host or security stack can change without warning, that turns an unreproducible complaint into a diagnosis.

SECTION 08

The bug that only broke for real visitors.

The membership form worked perfectly in testing and failed intermittently in production with "Security check failed" — and refreshing the page did not help.

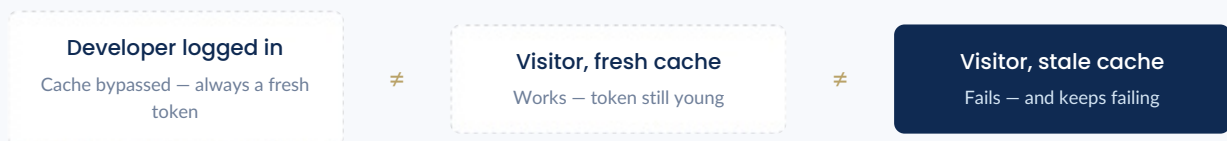
This is the most instructive problem on the project, because the cause sits at the seam between two things that are individually correct: WordPress security tokens, and full-page caching.

What was happening

WordPress protects form submissions with a nonce — a one-time token printed into the page and verified on submission. Nonces expire after roughly 24 hours. That is fine on an uncached site, because every visitor gets a freshly generated page with a freshly generated token.

On a cached site it is not fine. The page HTML is generated once and served to thousands of visitors for hours or days. The token is generated once too — and frozen inside that cached HTML. Once the cache entry passes the expiry window, every visitor served that page receives a token that is already dead. The form fails. And refreshing does nothing, because the refresh serves the same cached page carrying the same dead token.

WHY THE FAILURE WAS INVISIBLE IN TESTING



Logged-in users bypass page caching, so the form worked every time it was tested from the admin. It also worked for the first visitors after any cache purge. Only the third case failed — and that case is the majority of real traffic.

THE COMPOUNDING FACTOR

Caching plugins also combine and minify JavaScript into bundled files. Because the token is passed to the script through WordPress's script-localisation mechanism, it can end up frozen inside a cached *JavaScript bundle* as well as inside cached HTML — so purging the page cache alone does not necessarily clear it.

SECTION 08 · CONTINUED

Minting the token at submit time.

The fix was to stop trusting any token that came from the page, and instead request a freshly minted one from the server at the moment of submission.

```
// The nonce printed into the page gets frozen inside cached HTML.
// This endpoint hands the browser a freshly minted nonce at submit time.
function cla_ajax_refresh_nonce() {
    while ( ob_get_level() ) ob_end_clean();
    nocache_headers(); // must never be cached by a CDN / reverse proxy
    header( 'Content-Type: application/json; charset=utf-8' );
    echo wp_json_encode( [
        'success' => true,
        'data' => [
            'member' => wp_create_nonce( 'cla_member_nonce' ),
            'subscriber' => wp_create_nonce( 'cla_subscriber_nonce' ),
        ],
    ] );
    exit;
}
```

includes/ajax.php — the live nonce endpoint, registered for both logged-in and logged-out visitors

Three details that make it actually work

<code>nocache_headers()</code>	The endpoint that exists to defeat caching must itself never be cached. Without this the CDN would cache the response and serve one frozen token to everyone — recreating the original bug one layer up.
Output buffer cleared	Any stray output from another plugin would corrupt the JSON payload. The buffer is unwound before a single byte is written, and the content type is set explicitly.
A cache-busting parameter	The browser request carries a timestamp, so intermediate proxies cannot serve a stored copy of the response.

And a silent retry, for the case where it still fails

A failed nonce check returns a machine-readable `bad_nonce` code rather than only a human message. The front end recognises that code, fetches a brand new token, and resubmits once — silently. The visitor never sees an error at all. If the retry also fails, the human-readable message is shown.

```
// Stale nonce — fetch a brand new one and retry once,
// silently, so the visitor never sees the error.
if ( res && !res.success && res.data.code === 'bad_nonce' && !isRetry ) {
    getFreshNonce(opts.type).done(function(freshNonce){
        send(freshNonce, true);
    });
    return;
}
```

public/js/forms.js — one retry, guarded by a flag so it can never loop

Rejected tokens are also written to the error log with the originating IP, so the pattern is visible in the logs if it ever recurs rather than being silently absorbed.

SECTION 09

What the nonce was quietly protecting.

A nonce is not only a CSRF token — in practice it also raises the cost of automated submission. Handing out fresh tokens on request removes that friction, so it had to be replaced deliberately.

This is the part of the fix that is easy to skip. Once any client can ask for a valid token, a script can ask for one too. Three cache-safe controls were added to cover the gap.

01

Honeypot field

A hidden input no human ever sees or fills. A populated value is a definitive signal, and it costs a legitimate visitor nothing.

02

Elapsed-time check

Submissions completed under two seconds after the form became interactive are rejected. Nobody types a nine-field application that fast.

03

Per-IP throttle

Six submissions per address per ten-minute window, held in transients, applied separately to each form.

```
// The elapsed timer is measured in the browser on purpose – a
// server-rendered timestamp would itself be frozen in cached HTML.
function cla_is_probable_spam() {
    if ( ! empty( $_POST['cla_hp_field'] ) ) {
        return true; // hidden field only a bot would fill
    }
    $elapsed = isset( $_POST['cla_elapsed'] ) ? absint( $_POST['cla_elapsed'] ) : 0;
    if ( $elapsed > 0 && $elapsed < 2000 ) {
        return true; // submitted under 2 seconds after becoming interactive
    }
    return false;
}
```

includes/ajax.php — both checks are cache-safe by construction

THE DETAIL THAT TIES IT BACK TO THE ORIGINAL BUG

The elapsed timer is measured **in the browser**, not on the server. A server-rendered timestamp would have been frozen inside the cached HTML exactly like the token was — and every visitor would have appeared to take hours to fill in the form. The same failure mode, reintroduced by the fix. Measuring from the moment the script initialises in the visitor's own browser is the only version that survives caching.

Rejected submissions return a deliberately vague message. Telling an automated client which specific check it failed is free information for whoever is writing it.

SECTION 10

Three states, and an email at every edge.

The workflow is small, which is the point. Every transition is explicit, reversible from the admin, and carries its own correspondence.



What happens on approval

Approving a member is a single AJAX action from the members screen. It verifies the administrator's nonce, checks the `manage_options` capability separately, validates that the requested status is one of exactly two permitted values, updates the row, re-reads the member, and sends the matching template — approval or rejection — to that member's address. The response confirms both that the status changed and that the notification went out.

Only two target states accepted	The endpoint validates the requested status against an explicit list. An arbitrary string cannot be written into the status column by crafting a request.
Email failure cannot roll back approval	The send is wrapped in its own try/catch. If mail delivery fails, the member is still approved and the failure is logged — rather than leaving the record in an inconsistent state because an SMTP host was briefly unreachable.
Bulk actions exist for volume	Approve, reject and delete can be applied to a multi-selection from the members list, for the case where a batch of applications arrives at once.

Managing the list

The members screen carries free-text search across first name, family name and email; filters for state and status; paginated results at twenty-five per page; and CSV export that respects whatever filters are currently applied — so exporting "active members in Victoria" is a two-click operation rather than a spreadsheet exercise afterwards.

DEVELOPMENT CONTEXT

All list queries are built with prepared statements. Search terms are escaped through `esc_like()` before being wrapped in wildcards, so a percent sign typed into the search box is treated as a literal character rather than as a pattern.

SECTION 11

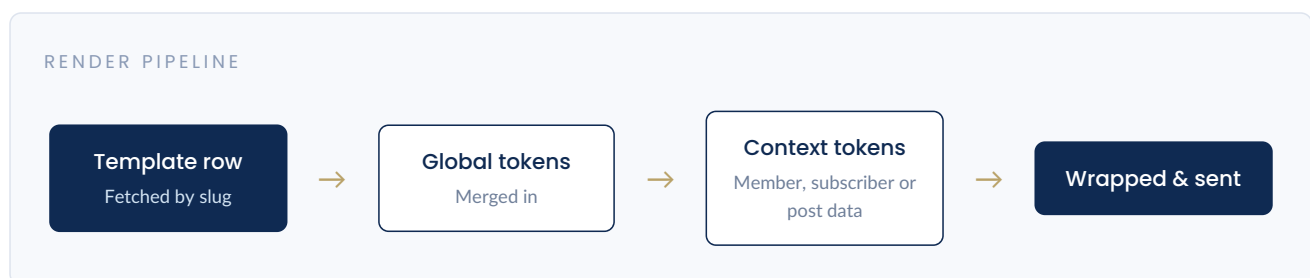
Seven templates the client can edit.

Every message the system sends is a database row with a stable slug, a subject and an HTML body — not a string buried in PHP.

SLUG	WHEN IT FIRES
member_user_confirmation	Application received — sent to the applicant, restating their submitted details
member_admin_notification	Application received — sent to the administrator, with a direct link to the members screen
member_approved	An administrator approves the application
member_rejected	An administrator rejects the application
subscriber_user_confirmation	Newsletter subscription confirmed to the subscriber
subscriber_admin_notification	New subscriber notified to the administrator
newsletter	A post is published — sent to the entire list

The token system

Templates contain double-brace placeholders that are resolved at send time. Two sets are merged: global tokens available everywhere — logo URL, site name, site URL, footer text, unsubscribe URL — and context tokens supplied by whichever routine is sending.



A SMALL BUT DELIBERATE ASYMMETRY

Tokens substituted into the **subject line** are stripped of all tags; tokens substituted into the **body** are not. That is not an oversight. Subject lines must be plain text, and allowing markup there would produce broken headers. The body legitimately receives composed HTML — the newsletter's featured image and social icon row are both passed in as markup fragments.

SECTION 11 • CONTINUED

Designed for email clients, not for browsers.

These are not plain notification emails. Each template is a full branded layout — a navy header with a gold accent rule, a status badge, a structured detail table, a call-to-action panel and a footer with website and unsubscribe links.

Building that to render correctly across desktop clients, webmail and mobile means abandoning modern CSS entirely. Every layout in these templates is built from nested tables with inline styles, because that is what email clients reliably support.

Constraints designed around

- Table-based layout throughout — no flexbox, no grid
- Inline styles on every element
- Fixed 600px content width with a max-width fallback
- Logo with an inline error fallback, so a broken image URL does not leave a gap
- Social icons as styled table cells rather than image files

Two wrapper strategies

Templates that carry their own header and footer are wrapped only in an outer shell — the page background, the centred card and the disclosure line.

Ad-hoc emails composed on the Send Email screen get the *full wrapper*: gold rule, navy logo header, padded content area and branded footer are all added around whatever the administrator typed.

Logo resolution has a fallback chain	An explicitly configured logo URL is used first; failing that, the site icon; failing that, the site name as styled text. The email header is never empty.
Social icons render conditionally	Each network appears only if a URL has been configured. With none set, the entire block returns empty rather than leaving a row of dead links.
Every template carries an unsubscribe link	Resolved from a global token, so it is present in every message without the client having to remember to include it.

The templates are seeded on activation and re-checked whenever the stored version differs from the plugin version, so a fresh install and an upgraded install converge on the same state.

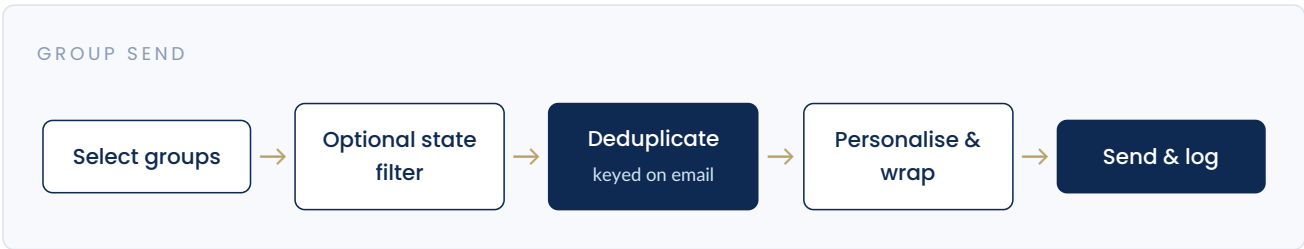
SECTION 12

Sending to everyone, and sending automatically.

Two dispatch paths — one an administrator triggers deliberately, one that fires the moment an article is published.

Group email

The Send Email screen opens with live counts of active members and subscribers, so the administrator knows the size of the audience before composing. Recipients are selected by group — members, subscribers, or both — with an optional state filter applied to the member side. Only members with active status are ever included; pending and rejected applicants are excluded by the query itself rather than by a checkbox somebody might forget.



Each message is personalised before sending — a name token is substituted per recipient, then the global tokens, then the whole thing is wrapped in the branded shell. Successes and failures are counted separately, and the send is written to the log with its group, subject, full body, both counts, the sending user and a timestamp.

The automatic newsletter

Publishing a post emails the entire list. The plugin hooks the post status transition and applies four guards before it will send anything.

Standard posts only	Pages and custom post types are ignored — which is what stops the forty-plus policy records from triggering a broadcast every time one is edited.
First publish only	The old status is checked. Editing an already-published article does not re-notify the list — the single most common failure mode in newsletter plugins.
Idempotency marker	A meta flag is written to the post before sending. If the transition hook fires twice, the second pass exits immediately.
Master switch	Automatic sending can be turned off in settings without deactivating the plugin.

Composition pulls the title, publication date, author display name, permalink and excerpt. Where no manual excerpt exists, one is generated from the content at fifty-five words. The featured image is rendered as an email-safe table with a fixed pixel width and a percentage fallback. Recipients are members and subscribers combined and deduplicated, and the run is logged like any other send — including the case where there were no recipients at all, which is logged explicitly rather than passing silently.

SECTION 13

Six screens, and no developer required.

Everything the organisation does week to week happens inside one WordPress menu. This section is the runbook.

CLA MEMBERSHIP MENU

Members

Subscribers

Send Email

Email Log

Email Templates

Settings

SCREEN	WHAT THE ADMINISTRATOR DOES HERE
Members	Review applications and approve or reject them, singly or in bulk. Search by name or email; filter by state or status; page through at twenty-five per row. Export to CSV with the current filters applied.
Subscribers	View and manage the newsletter list separately from the membership register.
Send Email	Compose and send to members, subscribers or both, with an optional filter by state. Live audience counts are shown before composing.
Email Log	Every send on record — group, subject, full body, delivered and failed counts, who sent it and when.
Email Templates	Edit the subject and body of all seven automated emails, using the published merge tags.
Settings	Notification address, sender name and address, logo, footer text, social links, success messages, the newsletter master switch, and database status with a repair action.

The three routine tasks

01

Review an application

Members → filter to Pending → Approve or Reject. The matching branded email is sent automatically. Nothing else to do.

02

Publish an article

Write the post, set a featured image, publish. The full list is emailed automatically — no separate send step.

03

Email a state

Send Email → tick Members → choose the state → compose → send. The result is logged automatically.

SECTION 13 • CONTINUED

Changing what the system says.

The most useful thing about a template engine is that the client never has to ask for a wording change. All seven automated emails are editable in the admin, and every value that appears in them is either a settings field or a merge tag.

Editing an automated email

1	Open Email Templates and choose the message — application confirmation, admin notification, approval, rejection, subscriber confirmation, subscriber notification, or newsletter.
2	Edit the subject line and the body. Merge tags in double braces are resolved at send time; the available tags are listed on the screen.
3	Save. The change applies to the next message sent — there is no publish step and no cache to clear.

Settings that change every email at once

Logo	One URL, used in the header of every template. Falls back to the site icon, then to the site name as styled text, so the header is never empty.
From name and address	Applied to every outbound message, automated and manual alike.
Notification address	Where new applications and new subscribers are reported. Changing it is one field, not a code edit.
Footer text	Appears at the bottom of every template.
Social links	Four optional URLs. Each icon renders only if its URL is set; with none set, the block disappears entirely rather than leaving dead links.
Success messages	What a visitor sees on screen after submitting either form — editable without touching the forms.
Newsletter switch	Turns automatic sending on or off without deactivating the plugin, for periods when articles should publish quietly.

If something looks wrong

Settings shows the live status of all four database tables as individual indicators. If any is missing, a single repair action recreates the structure and reseeds the templates. Both public forms also check for their table before writing, and if it is absent the visitor sees an instruction naming the exact screen and button that fixes it — rather than a broken form and a silent failure.

THE HANDOVER TEST

Adding a policy, publishing an article, reviewing an application, rewording a confirmation email, changing the notification address, emailing every member in one state, and checking what was sent last month — all seven are done by the client, in the admin, without a developer. That was the requirement, and it is the measure the build was designed against.

SECTION 14

Built to survive its own environment.

This plugin runs on a site whose host, caching layer and security stack can all change without notice. Several decisions exist purely to make that survivable.

Version-triggered migrations

The stored schema version is compared against the plugin version on every admin request. When they differ, table creation and template seeding re-run automatically — so a plugin update never leaves an installation on an older schema.

A repair button

Settings shows the live existence status of all four tables as individual indicators, with a one-click repair action. When something has gone wrong at the database level, the client can see it and fix it without a support ticket.

Table checks before writing

Both public submission handlers verify the target table exists before attempting an insert. A missing table produces a clear instruction naming the exact screen and button that fixes it — not a fatal error in front of a visitor.

Failure is contained

Every handler catches both exceptions and PHP errors, always responds with valid JSON, and unwinds the output buffer first. A fatal error inside an unrelated plugin cannot turn a form submission into a blank white screen.

Idempotent by design

Three separate mechanisms in the plugin are safe to run twice, and that is deliberate. Table creation uses `CREATE TABLE IF NOT EXISTS` through `dbDelta`. Template seeding checks for an existing slug before inserting. Form submissions update the existing row when the email address is already present rather than creating a duplicate. Nothing in the plugin breaks if it runs a second time — which is exactly the property you want when the trigger is a hook you do not fully control.

WHY THIS MATTERS MORE THAN IT SOUNDS

Most of these decisions cost a few lines each and are invisible when everything works. They exist because the alternative is a client emailing at 2am to say the form is broken, with no way to diagnose it remotely. A plugin that reports its own state clearly and can repair itself is worth considerably more to a small organisation than one that is merely correct.

SECTION 15

Challenges we faced — how we solved them.

Four real problems from this build, each taken from the symptom through to the outcome.

CHALLENGE 01

A security token frozen inside cached HTML

PROBLEM	The membership form failed intermittently in production with a security error, and refreshing the page did not fix it. It never failed in testing.
ANALYSIS	WordPress nonces expire after roughly 24 hours; full-page caching serves the same generated HTML for far longer. The token was being frozen inside the cache and outliving its own validity. Logged-in testing bypassed the cache entirely, which is why it looked fine. Minification made it worse by freezing the token inside a bundled script as well.
IMPLEMENTATION	A dedicated endpoint that mints a fresh token on request, explicitly marked uncacheable, with the output buffer unwound and the content type set before writing. The front end requests one immediately before every submission and treats the page-embedded token as a fallback only.
SOLUTION	A machine-readable failure code lets the front end fetch a new token and retry once, silently. The visitor never sees the error. Rejections are logged with the originating IP so the pattern stays visible.
OUTCOME	The form works regardless of how aggressively the site is cached, and remains compatible with any caching or CDN configuration the client's host adopts later.

CHALLENGE 02

The fix removed a protection nobody had named

PROBLEM	Once any client can request a valid token on demand, a script can request one too. The nonce had been doing quiet anti-automation work that the fix removed.
ANALYSIS	Every conventional replacement had the same flaw as the original: a server-rendered timestamp, a session marker or a per-page token would all be frozen in cached HTML exactly like the nonce was. Any control had to be cache-safe by construction.
IMPLEMENTATION	Three layers — a honeypot field invisible to humans, an elapsed-time check measured in the visitor's own browser from the moment the script initialises, and a per-IP transient throttle applied separately to each form.
SOLUTION	None of the three depends on server-rendered page content, so none can be defeated by caching. Rejections return a deliberately vague message rather than naming the check that failed.
OUTCOME	The protection the nonce provided incidentally is now provided deliberately, by controls designed for a cached environment rather than inherited from an uncached one.

SECTION 15 • CONTINUED

Challenges 03 and 04.

CHALLENGE 03

Branded email that survives real email clients

PROBLEM	The client wanted correspondence that carried the site's identity — branded header, status badge, structured detail table, call-to-action panel, footer — not the plain text most plugins emit. Email clients support almost none of the CSS that would make that straightforward.
ANALYSIS	Modern layout techniques are unusable here. Older desktop clients and several webmail providers still require table-based layout with inline styles, and gradient or rounded-corner effects degrade unpredictably across them.
IMPLEMENTATION	Every template built from nested tables with inline styles, at a fixed content width with a percentage fallback. Two wrapper strategies: an outer shell for templates carrying their own header and footer, and a full branded wrapper for ad-hoc emails composed in the admin.
SOLUTION	Fallback chains where assets can fail — configured logo, then site icon, then site name as text; social icons rendered as styled table cells rather than images, and omitted entirely when no URLs are configured.
OUTCOME	An applicant who submits a form on the website receives a confirmation that looks like it came from the same organisation, and degrades gracefully rather than breaking when an asset is missing.

CHALLENGE 04

Automatic sending that must never fire twice

PROBLEM	Publishing an article emails the entire list. The WordPress status transition hook can fire more than once in a single request, editing an already-published post passes through the same hook, and the site carries forty-plus policy records that are edited regularly.
ANALYSIS	A duplicate newsletter is not a cosmetic bug — for a membership organisation it damages the relationship with the list and harms sending reputation. A single condition on the new status is not sufficient protection.
IMPLEMENTATION	Four independent guards: post type restricted to standard posts so policy edits are excluded, the previous status checked so article updates are excluded, a per-post idempotency marker written before dispatch, and a master switch in settings.
SOLUTION	The idempotency marker is written <i>before</i> the send loop, not after. If anything fails mid-send, a retrigged hook still exits rather than restarting a partial broadcast from the beginning.
OUTCOME	Publishing is the only action required to reach the full list, and no combination of hook behaviour, policy editing or post updating can send the same article twice.

SECTION 16

The problems that no longer exist.

A website whose content can grow indefinitely, and a membership system that runs itself between decisions.

THE PROBLEM	HOW IT IS NOW SOLVED
A policy library that would outgrow its own navigation	Policies are structured records behind a paginated archive. Forty become eighty without a single template change, and the menu stays at six items.
Expansion needing a rebuild	The state framework is hierarchical from day one. Adding the remaining states is content entry against an architecture that already accounts for them.
Applications arriving as unstructured messages	Structured records with validated address, controlled state and explicit consent — deduplicated on email, searchable, filterable and exportable.
No difference between an applicant and a member	A three-state workflow with approve and reject actions, singly or in bulk, and rejected applications preserved on file rather than deleted.
Replies written by hand, inconsistently	Seven branded templates firing automatically at every transition, in the organisation's own identity, editable by the client without a developer.
No way to reach the list	Group sending to members, subscribers or both, filterable by state, deduplicated across groups, and permanently logged with the full body of what was sent.
Publishing reaching nobody	An article published is an article distributed. Four guards ensure it happens exactly once.
A form that failed silently for cached visitors	A submission path designed for a cached site, with cache-safe anti-automation controls replacing what the security token had been doing incidentally.
Dependence on a developer for routine work	Every weekly task — publishing, reviewing, emailing, rewording, reconfiguring — is done by the client in the WordPress admin.

The durable result

Both halves of the delivery share the same principle: put the complexity in the structure, so that routine work stays simple. Policies, articles and states are records rather than pages. Members, subscribers, templates and send history are tables the organisation owns. The plugin migrates its own schema, reports its own database state and can repair itself from a button.

IN SHORT

Classic Liberals Australia has a public platform that can carry an indefinitely growing body of policy, and a membership system that takes applications, routes them through human review, corresponds in the organisation's own voice at every step, and reaches the whole list the moment something is published — without anyone needing to remember to do it.

Web development that holds up **after** launch.

We build custom WordPress, Shopify, Laravel and Next.js platforms for clients across Europe, Australia, the US and India — with a bias toward systems a client's own team can run without us.

PROJECT

classicliberals.org.au

EMAIL

sales@rishfotech.com

STUDIO

www.rishfotech.com

PHONE

+91 98747 00540

STUDIO ADDRESS

Ecospace Business Park, Newtown, Kolkata, India