Manual de uso (Editor, Desarrollador y Mantenedor)

Este documento resume cómo utilizar, desarrollar y operar el cotizador. Si buscas instalación y despliegue paso a paso, revisa `DEPLOYMENT.md`.

- URL pública típica del frontend: `https://cotizador.aysafi.com`
- URL pública típica del backend: `https://emqx.aysafi.com`

Editor (uso diario de la app)

Objetivo: crear, revisar, aceptar o rechazar cotizaciones y enviar documentos por correo.

Ingreso y UI general

- Abre el sitio y verás el listado de cotizaciones más recientes.
- Los tiempos aparecen en español corto (ej.: “hace 1 min”, “ayer”, “en 3 h”).
- Cuando una cotización requiere atención, verás una etiqueta azul “Revisar”.
- La lista es compacta: nombre de cliente truncado, columnas alineadas y fechas normalizadas.

Crear y editar

- Completa los campos del formulario y guarda. El listado se actualiza en tiempo real.
- Si adjuntas logos, respeta el límite de tamaño (la app restringe a 5 MB). Los archivos quedan en `outputs/` del servidor.

Actualización en tiempo real

- La app usa eventos del servidor (SSE). Cuando hay cambios, el renglón parpadea sutilmente para indicar actualización.
- Si pierdes conexión brevemente (reinicios del servidor), se reconecta sola; no necesitas refrescar la página.

Aceptar o rechazar cotizaciones

- Aceptar: el botón “Aceptar” aparece cuando se cumplen las condiciones (p. ej., anticipo requerido en CLP según la lógica de negocio vigente).
- Rechazar: al rechazar, se solicita motivo opcional. Esto marca la cotización con “Revisar” para seguimiento posterior.
- Copiar: al usar “Copiar” como base de una nueva, la bandera “Revisar” se limpia.

Correos y PDFs

- Al guardar/actualizar o al aceptar/rechazar, la app puede enviar correos con HTML enriquecido.
- Los PDF incluyen marcas de agua condicionales según el estado (borrador, aceptada, rechazada) y el logo si existe.
- Si un correo falla, consulta al Mantenedor (ver sección “SMTP y correo”).

Página de diagnóstico (/admin/config)

- Página informativa que muestra la configuración efectiva que usa el frontend.
- No modifica nada; sirve para confirmar que el frontend apunta al backend correcto.
- El acceso administrativo requiere token cuando el backend así lo exige (ver Mantenedor → Seguridad).

Desarrollador (trabajo en local y contribución)

Objetivo: montar el entorno local, entender la estructura y agregar cambios de forma segura.

Estructura del repositorio (resumen)

- `backend/`: Express (API, SSE, emails, PDFs, uploads). `server.js` inicia HTTP/HTTPS.
- `frontend/`: React + Vite. Carga configuración en runtime desde `/config.js`.
- `backend/outputs/`: JSON, QR, PDFs y logos generados. En producción puede moverse fuera del repo o definirse con `OUTPUT_DIR`.
- `release/`: paquetes para subir a cPanel (generados por script).

Instalación y desarrollo

1. Instala dependencias en la raíz y frontend.

```
npm install
cd frontend
npm install
```

1. Arranca todo en dev (backend + frontend):

```
npm run dev
```

- El frontend corre en `http://localhost:5173` y se proxea al backend `http://localhost:5000` para `/api`, `/outputs` y `/config.js`.

Configuración en runtime (sin rebuild)

- El backend sirve `/config.js` con `window.__APP_CONFIG__ = { API_BASE, FRONTEND_URL }`.
- El frontend debe obtener URLs vía utilidades: `apiUrl('/ruta')` y `eventsUrl()`.
- Importante: `config.js` es script clásico; NO uses `export`. Ejemplo válido:

```
window.__APP_CONFIG__ = {
  API_BASE: 'https://emqx.aysafi.com',
  FRONTEND_URL: 'https://cotizador.aysafi.com'
};
```

Scripts útiles (raíz)

- `npm run dev`: backend + frontend en paralelo con recarga.
- `npm run backend`: solo backend con nodemon.
- `npm run frontend:dev`: solo frontend Vite.
- `npm run frontend:build`: build de producción (Vite).
- `npm run build:checked`: lint y build de frontend.
- `npm run test`: corre pruebas backend y frontend.
- `npm run package:cpanel`: prepara carpeta `release/cpanel-*/` con `frontend/dist`, `frontend/.htaccess` y `frontend/config.js.template`.

Pruebas y calidad

- Lint: `npm run lint` (JS/JSX).
- Tests backend: `npm run test:backend` (Node test runner).
- Tests frontend: `npm run test:frontend` (Vitest). Si ves warnings de React `act(...)`, revísalos pero no suelen bloquear.

Seguridad y smoke tests

- HTTPS local (recorrido completo de rutas y verificación básica de mixed content):

```
npm run test:security
```

- Producción (E2E): valida frontend y backend publicados. Salta pruebas de backend si no alcanza `:8443`.

```
# Overrides opcionales
$env:FRONTEND_URL_PROD="https://cotizador.aysafi.com";
$env:BACKEND_URL_PROD="https://emqx.aysafi.com:8443"; npm run
test:security:prod
```

Interpretación rápida:

- SKIP en backend: backend no alcanzable; revisa Nginx/certificados/firewall/servicio.
- 404 en `/admin/login`: falta SPA fallback (.htaccess en cPanel) → ver DEPLOYMENT.md.
- Advertencia sin CSP: aconsejado habilitar CSP en `.htaccess` para mitigar inyecciones.

Convenciones

- Ramas: usa `develop` para integrar y `main` para producción (ver `BRANCH_FLOW_QUICKSTART.md`).
- UI de tiempos: usa el formateador relativo ya incluido (español corto) para consistencia.
- Eventos en tiempo real: suscríbete vía `EventSource(eventsUrl())` con backoff (ya implementado).

- Al consumir la API: no codifiques URLs absolutas; usa `apiUrl()`.

Mantenedor (operación y soporte)

Objetivo: desplegar, mantener y resolver incidencias.

Despliegue

- Guía completa: `DEPLOYMENT.md`.
- Modos soportados:
  - Monolítico en cPanel (Node.js/Passenger) sirviendo también la SPA.
  - Separado: frontend estático en cPanel y backend en VPS con Nginx (recomendado) o HTTPS directo en Node.

Variables de entorno (backend)

- Esenciales: `JWT_SECRET`, `FRONTEND_URL`, `PUBLIC_API_BASE` (vacío si monolítico).
- Admin opcional: `ADMIN_PASSWORD` (protege diagnósticos y permite `POST /api/admin/login`).
- SMTP: `SMTP_HOST`, `SMTP_PORT`, `SMTP_USER`, `SMTP_PASS`.
- Logs HTTP: `MORGAN_FORMAT`.
- Salidas: `OUTPUTS_DIR` (si prefieres fuera del repo).
- HTTPS directo (opcional): `HTTPS=true`, `HTTPS_PORT=8443`, `TLS_CERT_FILE`, `TLS_KEY_FILE`, `TLS_CA_FILE`.

HTTPS y dominios

- Evita “Mixed Content”: si la página es `https://`, el backend también debe ser `https://`.
- Recomendado: Nginx en 443 con certificado y proxy a Node. Para pruebas, puedes usar HTTPS directo en Node.

Diagnóstico y seguridad

- `/admin/config`: visualiza la config efectiva del frontend y prueba `/api/config` del backend.
- Si `ADMIN_PASSWORD` está definido, obtén token con `POST /api/admin/login` y úsalo para endpoints/admin según corresponda.
- SSE: el backend expone `/api/events` (Content-Type: `text/event-stream`). Si hay proxy, desactiva buffering y amplía timeouts.

SMTP y correo

- Verificación: `npm run verify:smtp` y `npm run send:test-email`.
- Errores comunes: `ETIMEDOUT`, `Greeting never received` → revisa host/puerto/TLS y credenciales; valida puertos en el proveedor.

Archivos generados y backups

- PDFs/QR/JSON quedan en `outputs/` o en la ruta definida por `OUTPUTS_DIR`.

- Respáldalos periódicamente. Para regenerar PDFs: `npm run regenerate-pdfs` o uno puntual con `npm run regenerate-pdf`.

Problemas frecuentes (y soluciones)

- "Mixed Content bloqueado": corrige `API_BASE` a `https://` y reintenta.
- "config.js Unexpected token 'export'": reemplaza por script clásico con `window.__APP_CONFIG__`.
- "Conexión SSE inestable tras proxy": en Nginx usa `proxy_buffering off` y `proxy_read_timeout`/`proxy_send_timeout` altos.
- "No llegan correos": valida SMTP y puertos, revisa logs del backend (morgan y nodemailer).

Registro y monitoreo

- Revisa logs del proceso (systemd journal o consola de cPanel).
- Activa `MORGAN_FORMAT=combined` para trazabilidad en producción.

Inventario de rutas (API y frontend)

A continuación, todas las rutas relevantes del backend (API) y del frontend, con su propósito, autenticación y respuestas probables.

Backend (API)

- GET `/` → Salud del backend
  - Respuesta: `{ ok: true, message: 'Cotizador backend' }`
  - Uso: prueba rápida de que el servidor responde.
- GET `/config.js` → Configuración para el frontend (runtime)
  - Respuesta: JavaScript clásico que asigna `window.__APP_CONFIG__ = { API_BASE, FRONTEND_URL }`.
  - Cache: controlado por el servidor web; en producción se sirve como archivo dinámico.
- GET `/api/config` → Vista JSON de la config efectiva
  - Auth: si `ADMIN_PASSWORD` está definido, requiere header `Authorization: Bearer <token>` (obtenido en `/api/admin/login`).
  - 200: `{ API_BASE: '...', FRONTEND_URL: '...' }`
  - 401: `{ error: 'unauthorized' }` cuando hay admin configurado y falta/expiró el token.
- POST `/api/admin/login` → Login admin
  - Body JSON: `{ "password": "..." }`
  - 200: `{ ok: true, token }` (JWT, expira a 8h)
  - 401: `{ error: 'invalid_credentials' }`
  - 500: `{ error: 'admin_not_configured' }` si no hay `ADMIN_PASSWORD` definido en el backend.

- GET `/api/events` → SSE (event-stream) para actualizaciones en tiempo real
    - Headers: `Content-Type: text/event-stream`
    - Eventos: `ping`, `quote.created|updated|approved|rejected|needsReview|deleted`
    - Uso: el frontend crea `EventSource(eventsUrl())` y reconecta con backoff.
- Static `/outputs/*` → Archivos generados (solo lectura)
    - Ejemplos:
        - `/outputs/COT-2025-123456.json` (datos de una cotización)
        - `/outputs/pdfs/COT-2025-123456.pdf` (PDF de la cotización)
        - `/outputs/COT-2025-123456_qr.png` (QR)
- POST `/api/upload/logo` → Subir logo de empresa
    - Form-data: campo `logo` (imagen, máx. 5 MB)
    - 200: `{ ok: true, filename: 'logo_1699999999999.png' }`
    - 400: `{ error: 'No file uploaded' }`
    - 500: `{ error: 'Failed to save file' }`
- POST `/api/logs` → Ingesta de logs del frontend (dev)
    - Body JSON libre. Param opcional `?level=info|warn|error`.
    - 200: `{ ok: true }`

Rutas de empresa (`/api/empresa`):

- GET `/api/empresa` → Listado de empresas
    - 200: `[{ id, name, email, address, phone, taxId, logo, paymentDetails, terms, createdAt, updatedAt? }]`
- GET `/api/empresa/:id` → Empresa por id
    - 200: `{ ...empresa }`
    - 404: `{ error: 'not found' }`
- POST `/api/empresa` → Crear empresa
    - Auth: `Authorization: Bearer <token>`
    - Body: `{ name, email, address, phone, taxId, logo, paymentDetails, terms }`
    - 200: `{ ok: true, empresa }`
- PUT `/api/empresa/:id` → Actualizar empresa
    - Auth: `Authorization: Bearer <token>`
    - 200: `{ ok: true, empresa }`
    - 404: `{ error: 'not found' }`
- DELETE `/api/empresa/:id`
    - Auth: `Authorization: Bearer <token>`

- 200: `{ ok: true }`

Rutas de items (`/api/items`):

- GET `/api/items` → Lista de items (no auth)
  - 200: `[{ id, ... }]`
- POST `/api/items` → Crear item (no auth)
  - Body: libre (se añade `id`)
  - 200: `{ ok: true, item }`
- PUT `/api/items/:id` → Actualizar
  - 200: `{ ok: true, item }`
  - 404: `{ error: 'not found' }`
- DELETE `/api/items/:id` → Eliminar
  - 200: `{ ok: true }`

Rutas de cotizaciones (`/api/quotes`):

- GET `/api/quotes` → Lista-resumen
  - 200: `[{ file, quoteNumber, created_at, saved_at, token, client, total, currency, ... }]`
- GET `/api/quotes/next_ref` → Siguiente folio
  - 200: `{ next: 'COT-2025-123456' }`
- POST `/api/quotes` → Crear cotización
  - Body: `{ client, items:[...], total, currency?, isRequiredPrepayment?, prepaymentValue?, ... }`
  - Proceso: genera JSON/PDF, envía email al cliente, emite evento SSE
  - 200: `{ ok: true, file: 'COT-...json', token }`
  - 500: `{ error: 'server error' }`
- GET `/api/quotes/:file` → Obtener detalle
  - 200: `{ ...datos completos de la cotización... }`
  - 404: `{ error: 'not found' }`
- PUT `/api/quotes/:file` → Actualizar cotización
  - Body: datos de cotización (se preserva `quoteNumber` y `token`)
  - Efecto: reinicia estado de aprobación/rechazo, regenera PDF, reenvía email al cliente
  - 200: `{ ok: true, file, token }`
  - 500: `{ error: 'server error' }`

- DELETE `/api/quotes/:file` → Eliminar
  - Efecto: borra JSON, PDF y QR
  - 200: `{ ok: true }`
- POST `/api/quotes/:file/approve` → Aprobar o rechazar
  - Body:
    - Aceptar: `{ code6, approverName, prepayment? }`
    - Rechazar: `{ code6, reject: true, reason, approverName }`
  - Respuestas típicas:
    - `{ ok: true, rejected: true }` (rechazo confirmado)
    - `{ ok: true, needsReview: true }` (estaba rechazada; ahora requiere revisión interna)
    - `{ ok: true, regenerated: true }` (aprobada y PDF regenerado)
  - Errores comunes:
    - 400 `{ error: 'invalid code' | 'invalid prepayment' }`
    - 500 `{ error: 'approved_but_regen_failed' | 'review_regen_failed' | 'server error' }`

Frontend (React Router)

- `/` → Editor de cotización (QuoteEditor)
  - Crear/editar cotizaciones; la lista se actualiza en tiempo real.
- `/accept?file=COT-...json&token=...` → Vista de aceptación
  - Permite aceptar (con validación de prepago si aplica) o rechazar (motivo), usando el `token` o `code6`.
- `/admin/login` → Login administrador
  - Envía password a `/api/admin/login`; guarda `admin_token` (localStorage) y redirige.
- `/admin/company` → Mantenedor de empresa (protegido)
  - Requiere `admin_token`; usa `Authorization: Bearer <token>` contra `/api/empresa/*`.
- `/admin/config` → Visor de configuración (protegido)
  - Muestra `window.__APP_CONFIG__` y llama a `/api/config`.
- `/outputs/*` → Enlaces de descarga a PDFs/QRs desde la UI (servidos por el backend)
- `/config.js` → Cargado al inicio por el frontend para obtener `API_BASE`/`FRONTEND_URL`.

Ocultar 8443 detrás de 443 (HTTPS sin puerto visible)

Tienes dos formas de “ocultar” el puerto 8443 y ofrecer HTTPS estándar en 443:

Opción A (recomendada): TLS en Nginx (443) → backend HTTP

- Ejecuta el backend en HTTP (ej.: `PORT=5000`).
- Nginx escucha en 443 con certificado y hace `proxy_pass http://127.0.0.1:5000;`.
- Ventajas: configuración simple, no hay TLS doble.

Bloque Nginx:

```
server {
  listen 443 ssl http2;
  server_name emqx.aysafi.com;

  ssl_certificate     /etc/letsencrypt/live/emqx.aysafi.com/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/emqx.aysafi.com/privkey.pem;

  client_max_body_size 10m;

  location / {
    proxy_pass http://127.0.0.1:5000;
    proxy_http_version 1.1;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;

    # SSE
    proxy_read_timeout 3600s;
    proxy_send_timeout 3600s;
    proxy_buffering off;
  }
}
```

Opción B: TLS en Nginx (443) → backend HTTPS en 8443

- Mantén el backend con HTTPS propio en `8443`.
- Nginx hace `proxy_pass https://127.0.0.1:8443;` y reenvía tráfico cifrado.
- Útil si necesitas que el backend también sirva TLS por requerimientos internos.

Bloque Nginx (nota: upstream es HTTPS):

```
server {
  listen 443 ssl http2;
  server_name emqx.aysafi.com;

  ssl_certificate     /etc/letsencrypt/live/emqx.aysafi.com/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/emqx.aysafi.com/privkey.pem;

  location / {
    proxy_pass https://127.0.0.1:8443;
```

```
    proxy_http_version 1.1;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_read_timeout 3600s;
    proxy_send_timeout 3600s;
    proxy_buffering off;

    # Si el certificado interno no es de confianza, desactiva verificación
(solo si confías en la máquina)
    proxy_ssl_server_name on;
    proxy_ssl_verify off;
  }
}
```

Con cualquiera de las dos opciones, en el frontend usa `API_BASE = 'https://emqx.aysafi.com'` (sin puerto).

Despliegue automatizado “todo en 1” (VPS)

Guía rápida para levantar el backend en un VPS con systemd y Nginx en 443, apuntando al frontend con `config.js` (split hosting). Adáptalo a tu distribución.

1) Preparar servidor y clonar

```
sudo apt-get update -y && sudo apt-get install -y git curl ca-certificates
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt-get install -y nodejs nginx
sudo mkdir -p /opt/cotizador && sudo chown $USER:$USER /opt/cotizador
cd /opt/cotizador
git clone https://github.com/asdrubalfuentes/cotizador .
npm ci && (cd frontend && npm ci)
```

2) Crear `.env` del backend

```
FRONTEND_URL=https://cotizador.aysafi.com
PUBLIC_API_BASE=https://emqx.aysafi.com
JWT_SECRET=cambia-este-secreto
ADMIN_PASSWORD=clave-admin
SMTP_HOST=
SMTP_PORT=
SMTP_USER=
SMTP_PASS=
OUTPUTS_DIR=/var/lib/cotizador/outputs
MORGAN_FORMAT=combined
# Si usarás HTTPS directo en Node (opcional)
```

```
# HTTPS=true
# HTTPS_PORT=8443
# TLS_CERT_FILE=/etc/ssl/emqx/emqx.crt
# TLS_KEY_FILE=/etc/ssl/emqx/emqx_key.rsa
# TLS_CA_FILE=/etc/ssl/emqx/emqx.ca-bundle.crt
```

3) Crear directorio de salidas

```
sudo mkdir -p /var/lib/cotizador/outputs/pdfs /var/lib/cotizador/outputs/logos
sudo chown -R $USER:$USER /var/lib/cotizador
```

4) Servicio systemd

Archivo: `/etc/systemd/system/cotizador.service`

```
[Unit]
Description=Cotizador Backend
After=network.target

[Service]
Type=simple
WorkingDirectory=/opt/cotizador
Environment=NODE_ENV=production
ExecStart=/usr/bin/node backend/server.js
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
```

Activar:

```
sudo systemctl daemon-reload
sudo systemctl enable cotizador --now
```

5) Nginx 443 → backend

Archivo: `/etc/nginx/sites-available/cotizador-backend`

```
server {
  listen 80;
  server_name emqx.aysafi.com;
  location /.well-known/acme-challenge/ { root /var/www/html; }
```

```
  location / { return 301 https://$host$request_uri; }
}

server {
  listen 443 ssl http2;
  server_name emqx.aysafi.com;
  ssl_certificate     /etc/letsencrypt/live/emqx.aysafi.com/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/emqx.aysafi.com/privkey.pem;
  client_max_body_size 10m;
  location / {
    proxy_pass http://127.0.0.1:5000;
    proxy_http_version 1.1;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_read_timeout 3600s;
    proxy_send_timeout 3600s;
    proxy_buffering off;
  }
}
```

Habilitar y recargar:

```
sudo ln -s /etc/nginx/sites-available/cotizador-backend /etc/nginx/sites-
enabled/ || true
sudo nginx -t && sudo systemctl reload nginx
```

6) Frontend en cPanel

- Sube `frontend/dist/` al docroot del subdominio.
- Crea `config.js` con:

```
window.__APP_CONFIG__ = {
  API_BASE: 'https://emqx.aysafi.com',
  FRONTEND_URL: 'https://cotizador.aysafi.com'
};
```

7) Pruebas de humo

- `curl -s https://emqx.aysafi.com/` → `{ ok: true, ... }`
- Abrir `https://cotizador.aysafi.com/` → consola `window.__APP_CONFIG__` correcto.
- `/admin/login` → obtener token; `/admin/config` → ver backend `/api/config`.

Flujo de ramas (resumen operativo)

- Base: `main` (producción), `develop` (integración).
- Nuevo trabajo: `git checkout -b feature/nombre` desde `develop`.
- Al terminar: PR de `feature/*` → `develop`; squash/merge.
- Promocionar a prod: PR de `develop` → `main` tras lint/tests/build OK.
- Hotfix urgente: `git checkout -b hotfix/xyz main` → PR a `main` y back-merge a `develop`.

Referencias rápidas

- Despliegue detallado: `DEPLOYMENT.md`.
- Flujo de ramas: `BRANCH_FLOW_QUICKSTART.md`.
- Paquetes de release para cPanel: `npm run package:cpanel`.
- Diagnóstico de config: visita `/admin/config`.