MANUAL TÉCNICO - REPORTES+

Documentación técnica para mantenimiento, despliegue, arquitectura y soporte evolutivo

Código documental	RP-MT-003
Versión	3.0
Fecha	Junio 2026
Área responsable	Unidad de Informática HNJRE
Audiencia	Unidad de Informática, desarrollo, infraestructura y soporte técnico

Documento final para entrega institucional

Control del documento

Campo	Detalle
Título	Manual Técnico - Reportes+
Versión	3.0
Fecha de actualización	24/06/2026
Audiencia	Unidad de Informática, desarrollo, infraestructura y soporte técnico
Estado	Final para entrega
Marca vigente	Reportes+

Historial de cambios

Versión	Fecha	Descripción
3.0	24/06/2026	Actualización funcional y técnica del contenido en Markdown para reflejar el estado vigente de Reportes+.
3.0-F	26/06/2026	Consolidación en formato Word profesional, portada con logo actualizado, control documental e ilustraciones de apoyo.

Nota de actualización

Este documento sustituye la documentación técnica desactualizada de HN Soporte e incorpora la marca Reportes+, arquitectura vigente, APIs, seguridad, correo, notificaciones y anexo de glosario técnico.

Contenido

Ilustraciones de referencia

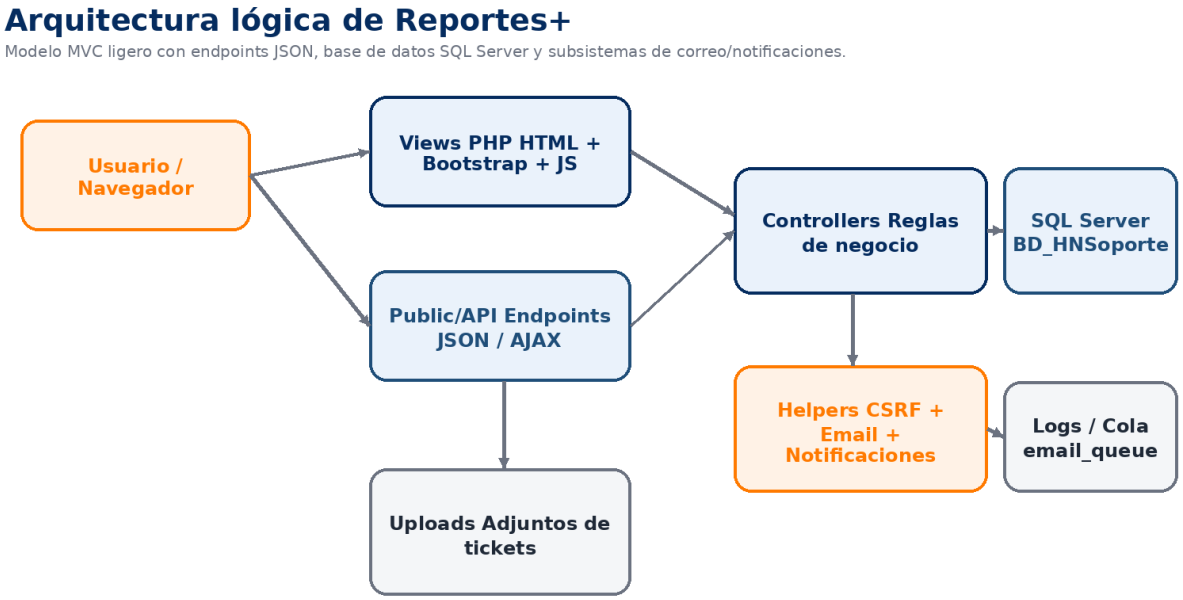

Figura 1. Arquitectura lógica de Reportes+.

Figura 2. Flujo técnico resumido del ticket.

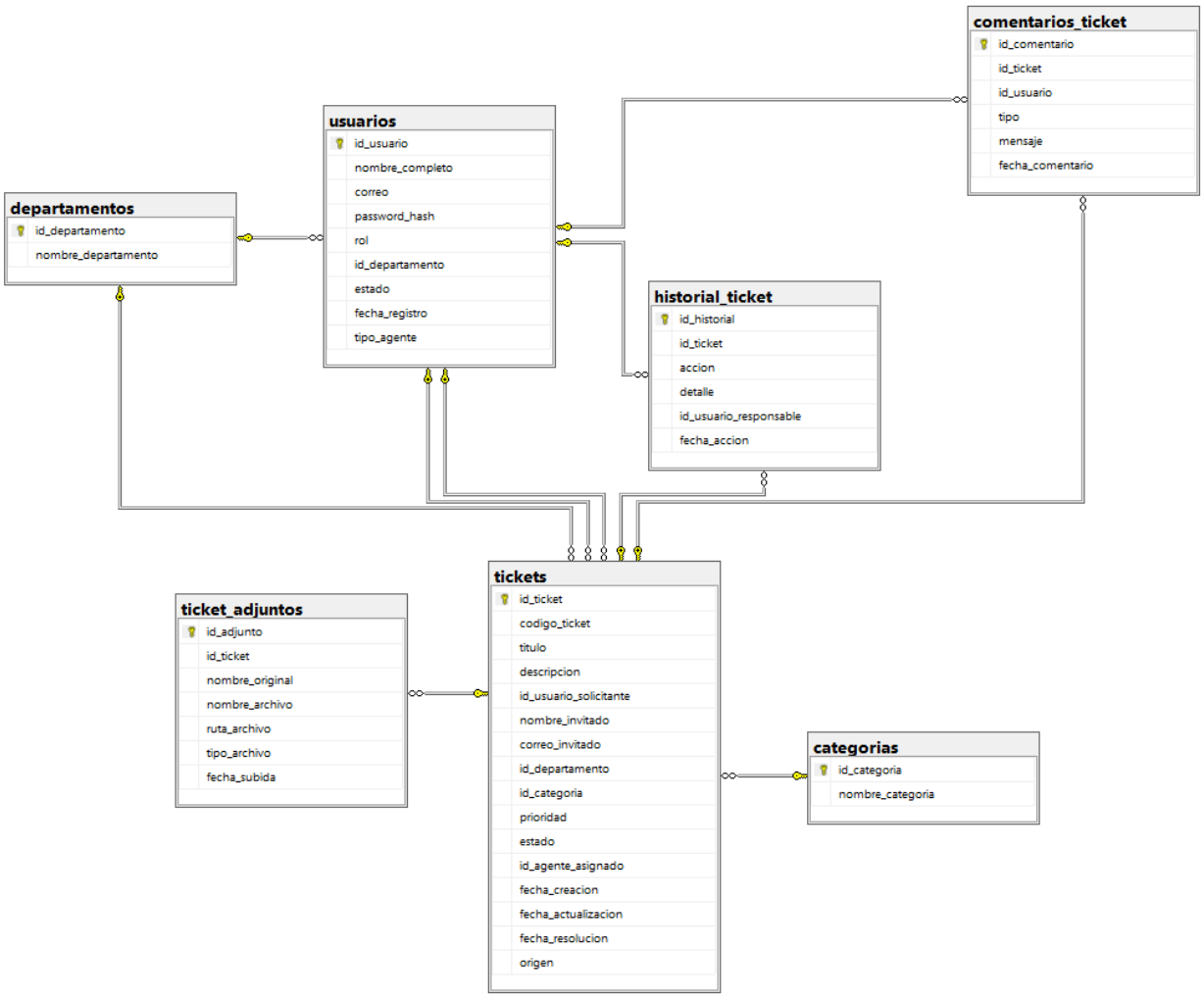

Figura 3. Diagrama de base de datos mantenido desde la documentación técnica anterior.

Documento técnico actualizado el **24/06/2026** para reflejar el estado real del repositorio y sustituir la documentación técnica desactualizada de HN Soporte v2.0.

1. Resumen ejecutivo

Reportes+ es una aplicación web de tickets y soporte construida en PHP nativo sobre SQL Server. Su arquitectura es ligera, sin framework full-stack, y combina:

- backend procedural y orientado a objetos en PHP,
- vistas server-rendered,
- endpoints JSON para interacciones dinámicas,
- cola de correos basada en archivos,
- notificaciones internas persistidas en base de datos,
- y reglas de negocio centradas en tickets, departamentos, agentes y auditoría.

Aunque la marca visible es **Reportes+**, varias rutas, nombres de carpeta y nombre de base de datos siguen usando el legado `hn_soporte` o `BD_HNSoporte` por compatibilidad técnica.

2. Objetivo técnico del sistema

El sistema busca resolver tres necesidades operativas:

1. Registrar solicitudes de soporte de forma estandarizada.
2. Orquestar su atención por agentes de soporte o programacion.
3. Medir volumen, tiempos y calidad del servicio.

3. Stack tecnologico y por que se usa

Backend

- **PHP nativo 8.x**
  - Se usa por simplicidad de despliegue en XAMPP y bajo costo operativo.
  - Permite mezclar render server-side con endpoints AJAX sin framework.
- **PDO + SQLSRV**
  - Se usa para acceder a SQL Server mediante consultas preparadas.
  - Reduce riesgo de SQL injection y mantiene compatibilidad con Microsoft SQL Server.

Base de datos

- **Microsoft SQL Server**
  - Elegido por el entorno institucional existente.
  - Se aprovechan funciones como `GETDATE()`, `DATEDIFF()`, `DATEPART()`, `STRING_AGG()` y restricciones `CHECK`.

Frontend

- **Bootstrap 5**
  - Maquetación responsive y componentes rapidos de UI.
- **JavaScript nativo + Fetch API**
  - Interacciones asincronas sin recargar pagina.
  - Uso fuerte en Kanban, notificaciones, dashboard, comentarios y reportes.
- **jQuery**
  - Se mantiene como apoyo para plugins heredados y algunos eventos de UI.
- **Select2**

  - Mejora selectores largos de departamentos, categorías y agentes.
- **SweetAlert2**
  - Confirmaciones y alertas visuales más claras que `alert()` o `confirm()`.
- **Chart.js**
  - Gráficas del dashboard general y del panel de usuario.
- **DataTables**
  - Búsqueda, paginación y ordenamiento de tablas de tickets.
- **SheetJS / XLSX**
  - Exportación de datos a Excel directamente desde el navegador.
- **Bootstrap Icons + Animate.css**
  - Apoyo visual sin depender de assets complejos.

Correo y notificaciones

- **SMTP implementado manualmente**
  - `helpers/EmailHelper.php` abre sockets SMTP y maneja `EHLO`, `STARTTLS`, `AUTH LOGIN/PLAIN`, `DATA` y `QUIT`.
  - No depende de PHPMailer ni Composer.
- **Notification API del navegador**
  - Notificaciones de escritorio cuando el navegador tiene permiso.

4. Arquitectura general

La aplicación sigue un MVC ligero y pragmatico:

- **Config**
  - Inicializa conexion, branding y correo.
- **Controllers**
  - Contienen la lógica de negocio.
- **Views**
  - Renderizan HTML y disparan `fetch()` hacia endpoints.
- **Public/API**
  - Expone endpoints JSON para operaciones asincronas.
- **Helpers**
  - Resuelven preocupaciones transversales como CSRF y email.

No hay ORM, no hay service container y no hay sistema de rutas avanzado. La navegacion depende de archivos PHP accesibles por ruta directa.

5. Estructura de carpetas y que hace cada una

```
config/
  branding.php           Marca visible del sistema y URLs de assets
  db.php                 Conexion PDO, configuración SQLSRV y auto-migraciones ligeras
  mail.php               Parámetros base para transporte SMTP
  secrets.local.php      Credenciales reales del entorno (no documentar valores)

controllers/
  AuthController.php         Login, logout y sesión
  TicketController.php       Nucleo funcional del sistema
  UserController.php         CRUD de usuarios y preferencias de correo
  ApiController.php          KPIs, series, reportes y agregaciones
  NotificationHelper.php     Notificaciones internas persistidas en BD
```

```
  CategoryController.php     CRUD de categorías
  DepartmentController.php   CRUD de departamentos

helpers/
  Csrf.php                Generación y validación de token CSRF
  EmailHelper.php         Cola, log y envío SMTP/manual de correos

logs/
  email.log               Historial de entregas y eventos de correo
  email_queue/            Cola de correos en JSON

public/
  index.php               Redirige al home según rol
  login.php               Acceso al sistema
  logout.php              Cierre de sesión
  register.php            Solicitud de cuenta
  guest_ticket.php        Creación publica de tickets
  check_ticket.php        Consulta publica por código
  uploads/tickets/        Archivos adjuntos
  api/                    Endpoints AJAX / JSON

views/
  layout/                 Header, sidebar, scripts globales y seguridad frontend
  tickets/                CRUD operativo del ticket
  reports/                Reportería y exportaciones
  users/                  Gestión de usuarios
  guest_users/            Aprobacion de invitados
  admin/                  Bitácora de correos
  dashboard.php           Dashboard gerencial
  user_dashboard.php      Panel personal del usuario
  profile.php             Perfil y preferencias de correo

sql/
  schema.sql              Estructura base inicial
  update_attachments.sql Migración manual de adjuntos
  add_notificaciones.sql Migración manual de notificaciones

Documentos/
  Manuales funcionales y técnicos versionados en Markdown
```

6. Flujo principal del ticket

6.1 Creación autenticada

Archivo principal:

- `views/tickets/create.php`

Lógica central:

- `TicketController::createTicket()`

Proceso:

1. Valida título, descripción, departamento, categoría y prioridad.
2. Valida adjuntos por extensión, MIME y tamaño.
3. Inserta en `tickets`.
4. Si hay agente inicial, inserta en `ticket_agentes`.
5. Guarda adjuntos en `public/uploads/tickets/`.
6. Registra auditoría.

7. Genera notificación interna.
8. Encola correos al solicitante y al agente según reglas y preferencias.

6.2 Creación como invitado

Archivo:

- `public/guest_ticket.php`

Particularidades:

- `id_usuario_solicitante = NULL`
- `origen = 'invitado'`
- se usa `nombre_invitado` y `correo_invitado`
- prioridad inicial fija en `Media`

6.3 Código del ticket

Algoritmo actual:

- prefijo fijo `HD-`
- concatenado con `strtoupper(substr(md5(uniqid()), 0, 6))`

Ejemplo:

- `HD-A54E22`

7. Asignación, agentes y cambios de estado

La plataforma ya no depende solo de `id_agente_asignado`.

Modelo actual:

- columna legado `tickets.id_agente_asignado`
- tabla puente `ticket_agentes`

Metodo principal:

- `TicketController::assignTicketDetailed()`

Reglas:

- Si se agregan agentes y el ticket estaba `Nuevo` o `En_Espera`, pasa a `En_Progreso`.
- Si se quitan todos los agentes y estaba `Nuevo` o `En_Progreso`, pasa a `En_Espera`.
- Solo se notifican por correo los agentes realmente nuevos en la asignación.
- El solicitante puede recibir aviso de asignación según sus preferencias.

Regla para resolver

Antes de cambiar a `Resuelto`, `views/tickets/update_status.php` valida que:

- exista al menos un agente asignado
- exista al menos un comentario hecho por alguno de los agentes asignados

8. Comentarios y conversacion

Endpoint principal:

- `public/api/add_comment.php`

Tipos:

- `público`
- `interno`

Regla de visibilidad:

- `admin` y `agente` pueden ver ambos.
- `usuario` solo ve comentarios publicos.

Seguridad:

- valida sesión
- valida CSRF
- valida acceso con `TicketController::canSessionAccessTicket()`
- impide comentar tickets `Cerrado`

Correos derivados:

- a agentes asignados por mensajes nuevos en tickets propios
- al solicitante cuando soporte comenta
- siempre filtrados por preferencias de correo del usuario

9. CSAT y calificación por estrellas

Endpoint:

- `public/api/save_csat.php`

Reglas:

- solo tickets `Resuelto` o `Cerrado`
- solo una calificación por ticket
- solo el solicitante original puede calificar
- se guarda en:
  - `tickets.csat_estrellas`
  - `tickets.csat_comentario`
  - `tickets.csat_fecha`

Visualización:

- `views/tickets/view.php`
- `views/dashboard.php`
- `controllers/ApiController.php`

10. Notificaciones internas

Componente principal:

- `controllers/NotificationHelper.php`

Persistencia:

- tabla `notificaciones`

Casos principales:

- `notifyNewTicket()`
  - notifica a admins y agentes activos cuando se crea un ticket
- `notifyNewMessage()`
  - si comenta un usuario, notifica a agentes asignados y admins
  - si comenta soporte, notifica al solicitante registrado

Consumo:

- `public/api/get_notifications.php`
- `public/api/mark_read.php`
- `public/api/check_notifications.php`

- scripts en `views/layout/header.php`

11. Subsystema de correo

11.1 Componente central

- `helpers/EmailHelper.php`

Responsabilidades:

- construir plantilla HTML de correo
- encolar envios
- procesar cola
- enviar por SMTP
- registrar historial
- resolver branding y links
- normalizar texto para evitar problemas con acentos

11.2 Cola de correos

Directorio:

- `logs/email_queue/`

Formato:

- un JSON por correo encolado

Procesamiento:

- `EmailHelper::processQueue()`
- `EmailHelper::processQueueIfDue()`
- `public/api/process_email_queue.php`
- `process_email_queue.php`
- heartbeat desde `views/layout/header.php`

Reglas operativas actuales:

- el heartbeat intenta procesar la cola con enfriamiento de 30 segundos
- el endpoint AJAX procesa pocos correos por llamada para no bloquear la experiencia
- el script raiz permite procesamiento manual o agendado

11.3 Bitácora de correos

Vista:

- `views/admin/email_log.php`

Fuente:

- `logs/email.log`
- estado actual de `logs/email_queue/`

12. Paneles y reportería

12.1 Dashboard gerencial

Vista:

- `views/dashboard.php`

Endpoint:

- `public/api/dashboard_data.php`

Motor:

- `ApiController::getDashboardStats()`

Regla analítica importante:

- Los volumenes de entrada y pendientes usan `fecha_creacion`.
- Los resueltos, tiempos, ranking y CSAT usan `fecha_resolucion`.

Eso evita distorsionar KPIs cuando tickets viejos se resuelven en un período nuevo.

12.2 Panel del usuario

Vista:

- `views/user_dashboard.php`

Endpoint:

- `public/api/user_dashboard.php`

Incluye:

- KPIs del usuario
- tickets recientes
- gráfica por estado
- evolucion
- top solicitantes del departamento

12.3 Reportes

Vistas y endpoints:

- `views/reports/index.php`
- `public/api/stats.php`
- `public/api/report_agents.php`
- `public/api/report_detail.php`
- `public/api/ticket_info.php`
- `views/reports/export_excel.php`
- `views/reports/export_excel_consolidado.php`
- `views/reports/print_view.php`

13. Modelo de datos funcional

Tablas principales

- `usuarios`
  - usuarios del sistema, roles, departamento, estado, hash y preferencias de correo
- `departamentos`
  - catalogo de unidades solicitantes
- `categorías`
  - clasificacion funcional del ticket
- `tickets`
  - entidad central del sistema
- `ticket_agentes`
  - relacion muchos-a-muchos entre ticket y agente

- `comentarios_ticket`
  - historial conversacional
- `historial_ticket`
  - auditoría administrativa y operativa
- `ticket_adjuntos`
  - archivos asociados
- `notificaciones`
  - notificaciones internas

Columnas auxiliares relevantes agregadas por evolucion

- `tickets.csat_estrellas`
- `tickets.csat_comentario`
- `tickets.csat_fecha`
- columnas de preferencias de correo en `usuarios`

14. Seguridad aplicada

Credenciales

- `password_hash()` para almacenar contrasenas
- `password_verify()` para autenticar

Sesión

- `session_regenerate_id(true)` en login
- timeout de 4 horas si no está marcado `keep_logged_in`

CSRF

Clase:

- `helpers/Csrf.php`

Integracion:

- token en formularios
- meta tag
- wrapper global de `fetch()` en `views/layout/header.php`

SQL Injection

- uso predominante de `prepare()` + `execute()`
- parámetros enlazados en consultas sensibles

Control de acceso

Metodos clave:

- `TicketController::canSessionAccessTicket()`
- `TicketController::canSessionRateTicket()`

Reglas:

- `admin` y `agente` tienen acceso operativo total a tickets
- `usuario` queda restringido a tickets propios o del departamento permitido por la lógica del sistema
- endpoints críticos validan rol antes de ejecutar

Adjuntos

Validaciones en `TicketController`:

- extensión permitida
- MIME esperado
- tamaño máximo 10 MB
- `move_uploaded_file()`

15. Auto-migraciones

`config/db.php` ejecuta auto-migraciones ligeras al conectarse:

- columnas CSAT
- tabla `ticket_adjuntos`
- tabla `historial_ticket`
- columnas de preferencias de correo

Ventaja:

- agiliza despliegues pequenos

Riesgo:

- mezcla lógica de runtime con evolucion de esquema
- no reemplaza una estrategia formal de migraciones versionadas

16. APIs y endpoints principales

Ruta	Uso
public/api/dashboard_data.php	KPIs y series del dashboard general
public/api/user_dashboard.php	KPIs del panel del usuario
public/api/get_tickets.php	Fuente del Kanban
public/api/update_assignment.php	Asignación multiple de agentes
views/tickets/update_status.php	Cambio de estado desde detalle o Kanban
public/api/update_attributes.php	Cambio de prioridad, categoría u otros atributos
public/api/add_comment.php	Alta de comentario y adjuntos
public/api/chat_messages.php	Polling de nuevos mensajes
public/api/get_notifications.php	Bandeja de notificaciones
public/api/mark_read.php	Marcar notificaciones como leidas
public/api/check_notifications.php	Sondeo de actividad
public/api/process_email_queue.php	Consumo de cola de correos
public/api/save_csat.php	Guardar calificación
public/api/report_agents.php	Rendimiento por agente
public/api/report_detail.php	Detalle de tickets filtrados
public/api/stats.php	KPIs de reportes
public/api/ticket_info.php	Modal de detalle en reportes
public/api/register_account.php	Solicitud publica de cuenta

17. Parámetros y archivos de configuración

`config/branding.php`

Centraliza:

- nombre visible del sistema

- logos e iconos
- URL base
- video del login

Usa `filemtime()` para agregar versión a logo y video y reducir problemas de cache.

`config/db.php`

Centraliza:

- lectura de secretos
- construccion del DSN SQLSRV
- opciones PDO
- auto-migraciones

`config/mail.php`

Centraliza:

- transporte
- host
- puerto
- cifrado
- modo de autenticación
- remitente
- timeout
- simulacion

18. Mantenimiento operativo recomendado

- Mantener `logs/` y `public/uploads/tickets/` con permisos correctos de escritura.
- Monitorear `logs/email.log` y `views/admin/email_log.php`.
- Programar `process_email_queue.php` si se desea un respaldo adicional al heartbeat web.
- Respaldar la base de datos y la carpeta de adjuntos.
- No publicar `config/secrets.local.php`.

19. Riesgos técnicos y observaciones

- El proyecto combina rutas nuevas y legado, por lo que conviene documentar cambios antes de refactorizar.
- La aplicación no usa framework ni ORM; eso facilita despliegue, pero concentra mucha responsabilidad en pocos archivos grandes, especialmente `TicketController.php` y `views/layout/header.php`.
- Las consultas analiticas y de permisos deben revisarse con cuidado antes de tocar reportes, Kanban o acceso por departamento.

20. Conclusiones técnicas

Reportes+ es un sistema maduro para su contexto operativo, con varias mejoras funcionales ya integradas:

- marca visual centralizada
- asignación multiple de agentes
- correo con cola y bitácora
- CSAT
- proteccion CSRF
- preferencias personales de correo
- dashboards y reportes con mejor criterio temporal

La documentación técnica debe seguir tratandose como parte del producto, porque varias reglas criticas del sistema no son obvias a simple vista y estan distribuidas entre controladores, vistas y endpoints.

Anexo A. Glosario técnico complementario

Uso del anexo
Este glosario se incorpora como referencia complementaria para facilitar la lectura técnica y mantener compatibilidad documental con términos heredados del proyecto.

Glosario Técnico - Reportes+ (legado documental)

Actualizado el **24/06/2026**.

Nota importante

Este archivo se conserva como glosario complementario por compatibilidad historica, pero la referencia principal del proyecto ahora es:

- `Documentos/Manual_Tecnico_v2.0.md`

El sistema visible al usuario se llama **Reportes+**, aunque varias rutas internas, nombres de archivo y la base de datos sigan usando el nombre técnico legado `hn_soporte` o `BD_HNSoporte`.

Terminos base

API

Conjunto de endpoints HTTP usados por el frontend para leer o escribir información sin recargar toda la pagina. En este proyecto la mayoria vive en `public/api/`.

CSAT

Métrica de satisfaccion del usuario final. Se guarda en las columnas `csat_estrellas`, `csat_comentario` y `csat_fecha` de `tickets`.

CSRF

Mecanismo de defensa que obliga a que los formularios y peticiones `POST` incluyan un token valido de sesión. La implementación actual se encuentra en `helpers/Csrf.php`.

Dashboard

Vista gerencial o personal que resume KPIs, volumenes y tendencias de tickets mediante Chart.js.

Email queue

Cola de correos salientes basada en archivos JSON dentro de `logs/email_queue/`. Evita que cada acción del usuario espere el envío SMTP completo.

Historial de ticket

Registro de auditoría administrativa u operativa guardado en la tabla `historial_ticket`.

Invitado

Persona sin cuenta autenticada que puede crear ticket desde `public/guest_ticket.php` y consultar su código desde `public/check_ticket.php`.

Kanban

Vista operativa de tickets por estado, utilizada principalmente por administradores y agentes.

Notificación interna

Aviso guardado en la tabla `notificaciones` y mostrado en la campana del sistema.

Preferencias de correo

Conjunto de columnas en `usuarios` que permiten decidir que tipos de correo quiere recibir cada persona, según sea solicitante o agente asignado.

Reportes+

Nombre comercial actual de la plataforma. La configuración visual se resuelve desde `config/branding.php`.

SQLSRV / PDO_SQLSRV

Drivers de PHP necesarios para conectarse a Microsoft SQL Server usando PDO.

Ticket

Unidad central de trabajo del sistema. Representa una incidencia, solicitud o requerimiento y fluye por estados como `Nuevo`, `En_Progreso`, `En_Espera`, `Resuelto` y `Cerrado`.

Ticket_agentes

Tabla puente que materializa la asignación de varios agentes a un mismo ticket.