IFTS 18

PP3: Diseño y arquitectura de sistemas

# Documento general del proyecto: Ticketear

Grupo 4:

**Matías Gómez**

**Leandro Mendoza**

**Tomás Patriarca**

# Ticketear - Práctica Profesionalizante III

**Sistema:** Ticketear - Plataforma de tickets para PyMEs
**Empresa desarrolladora:** Sistemas Garín S.R.L.
**Modalidad:** Trabajo grupal (3 integrantes)
**Enfoque:** MVP viable en un cuatrimestre

## 1.1. Contexto

El proyecto sucede en el marco académico de la materia *Prácticas Profesionalizantes III* y simula un caso real donde la empresa de desarrollo **Sistemas Garín S.R.L.** ofrece un producto a PyMEs.

- **Lugar/Institución:** Empresas PyME del mercado local argentino que buscan mejorar la atención de sus clientes.

- **Características del entorno:** Organizaciones pequeñas o medianas (10–200 empleados), con recursos limitados para sistemas complejos y con procesos de soporte poco estructurados. Usan herramientas básicas como mail, hojas de cálculo o WhatsApp para gestionar incidencias.

## Problema o necesidad detectada

Las PyMEs reciben consultas y reclamos de sus clientes por múltiples canales, pero no tienen trazabilidad ni centralización. Esto genera:

- Tickets que se pierden o quedan sin respuesta.

- Dificultad para medir tiempos de resolución.

- Mala experiencia del cliente y pérdida de fidelización.

Se necesita un sistema simple y económico que permita **recibir tickets automáticamente** y darles seguimiento.

## 1.2. Tipo de Sistema27

El sistema sigue un modelo cliente-servidor:

1. **Nodo - Dispositivo del Usuario (Cliente):** Ejecuta el Navegador Web, que renderiza la interfaz (el Panel de Admin hecho en React o el Formulario Web de la PyME ).

2. **Nodo - Servidor de Aplicación (Nube):** Es el nodo central que aloja el backend. Su componente principal es la API de Ticketear (Node.js), que procesa toda la lógica de negocio y recibe las peticiones HTTPS.
3. **Nodo - Servidor de Base de Datos (Nube):** Es el nodo responsable de la persistencia de datos. Ejecuta el componente Base de Datos (PostgreSQL), al cual la API accede para leer y escribir.
4. **Nodo - Servidor de Email Externo (SaaS):** Es un servicio de un tercero (como SendGrid ). Contiene el Servicio SMTP, que es consumido por nuestra API para enviar las notificaciones por correo.

## 1.3. Actores Clave (Usuarios Involucrados)

- **Cliente de la PyME (final):** crea el ticket. Puede hacerlo de dos formas:

    1. **Web de la PyME:** la empresa coloca un formulario (nombre, apellido, mail, problema). El front lo desarrolla la PyME, nosotros proveemos los **endpoints de API** para que al enviarlo se cree el ticket en Ticketear.

    2. **Correo electrónico:** el cliente manda un mail a una dirección específica; el sistema lo convierte en ticket.

- **Administrador de la PyME:** usuario con acceso al ambiente asignado a su empresa. Puede loguearse, ver los tickets creados (por web o mail), asignarlos a responsables y dar seguimiento.

- **Agente de soporte (opcional):** en organizaciones con varios empleados, puede haber agentes que gestionen tickets asignados por el administrador.

## 1.4. Análisis FODA

**Fortalezas (internas, positivas)**

- Solución simple, concreta y alcanzable en un cuatrimestre.

- Diferenciación: integración **vía API** y **correo electrónico**, dos canales claros para crear tickets.

- Enfoque en PyMEs con necesidades reales, sin sobrecargar de features.

- Desarrollo en tecnologías conocidas (JavaScript/Node, React, PostgreSQL).

- Control total del backend: se entrega un ambiente por cliente (aislamiento).

**Oportunidades (externas, positivas)**

- Muchas PyMEs aún usan hojas de cálculo o mails sin trazabilidad → mercado potencial.
- Posibilidad de escalar en el futuro a más canales (WhatsApp, chatbots).
- Mercado local valora soluciones económicas y soporte en español.
- Crecimiento de software SaaS para PyMEs → chance de ofrecer suscripciones simples.

**Debilidades (internas, negativas)**

- Dependencia de servicios externos para correo entrante/saliente (ej. SMTP, SendGrid).
- Riesgo de problemas de seguridad si no se configuran bien los accesos por API/mail.
- No habrá funcionalidades avanzadas (SLA, reportes complejos, automatizaciones).

**Amenazas (externas, negativas)**

- Competidores establecidos (GLPI, Zendesk, Freshdesk) aunque con mayor complejidad.
- Posible resistencia de las PyMEs a integrar sus formularios con APIs externas.
- Dificultades técnicas en la captura de correos (dependencia de hosting/mailservers).
- Si el sistema no es confiable (caídas, tickets perdidos), la confianza se pierde rápido.

# 2. Objetivos

## 2.1. Objetivo General

Proveer una plataforma centralizada y multiempresa donde cada PyME pueda recibir tickets creados por sus clientes y gestionarlos en un panel web simple.

## 2.2. Objetivos Especificos

- Que los clientes de la PyME tengan un canal formal para registrar sus problemas.
- Que los administradores de la PyME visualicen y den seguimiento a los tickets en un solo lugar.
- Que se reduzca el tiempo de respuesta y se aumente la satisfacción de los clientes.

# 3. Alcance

## 3.1. Funcionalidades incluidas (MVP)

- Creación de tickets vía API (desde formularios externos) y vía email.
- Panel web para administradores con login, listado y detalle de tickets.
- Posibilidad de asignar tickets a agentes internos.
- Cambio de estado de ticket (abierto, en progreso, cerrado).
- Notificación por mail básica (confirmación de ticket creado).

## 3.2. Limitaciones (Fuera de alcance)

- Integración con WhatsApp, Telegram u otros canales.
- Funcionalidades avanzadas de automatización (reglas, SLA complejos).
- Multitenancy avanzado (se dará un ambiente por empresa, sin portal único).

- Reportes complejos o analítica avanzada (se limitará a listado básico de tickets).

- Base de conocimiento o portal de autoayuda para clientes finales.

# 4. Análisis de Usuarios y Experiencia de Usuario (UX)

## 4.1. Perfiles de Usuario (User Personas)

### Persona 1 – Cliente Final

- **Nombre ficticio:** Lucía Fernández
- **Edad y ocupación:** 29, diseñadora freelance
- **Objetivos:** Registrar reclamos fácilmente, recibir confirmación y hacer seguimiento.
- **Necesidades:** Formulario simple, notificaciones claras, acceso desde el celular.
- **Frustraciones:** No recibir respuesta, tener que repetir datos, tiempos largos de espera.
- **Contexto de uso:** Móvil (Android/iOS), 1–2 veces por mes, en movimiento o en su estudio.

### Persona 2 - Administrador PyME

- **Nombre ficticio:** Carla Medina
- **Edad y ocupación:** 41, dueña/gerenta de PyME
- **Objetivos:** Ver todos los tickets en un panel, asignarlos a su equipo, responder dudas de clientes.
- **Necesidades:** Listado centralizado, filtros por estado, métricas básicas, poder responder al cliente desde el sistema.
- **Frustraciones:** Tickets dispersos en múltiples canales, falta de reportes claros.
- **Contexto de uso:** Notebook en oficina/teletrabajo, revisa diariamente.

### Persona 3 - Agente de Soporte

- **Nombre ficticio:** Juan Ríos
- **Edad y ocupación:** 34, técnico de soporte

- **Objetivos:** Ver tickets asignados, actualizarlos, cerrarlos y enviar respuestas al cliente.
- **Necesidades:** Bandeja de “mis tickets”, filtros rápidos, poder redactar comentarios al cliente.
- **Frustraciones:** Tickets incompletos, poca visibilidad de prioridades.
- **Contexto de uso:** PC de escritorio, 6–8 horas diarias en oficina.

## 4.2. Historias de Usuario (Escenarios de Uso)

**HU1 - Cliente externo**
“Como *cliente de la PyME*, quiero enviar un correo electrónico describiendo mi problema para que *se genere un ticket automáticamente*, de modo que pueda recibir confirmación y seguimiento de mi caso.”

**HU2 - Administrador de PyME**
“Como *administrador de la PyME*, quiero ver un listado de todos los tickets creados en mi panel web para *asignarlos a mis agentes internos* y darles seguimiento.”

## 4.3. Principios de Diseño y Heurísticas

### Principios de Gestalt aplicados

- **Proximidad**: Los elementos relacionados (datos del cliente, estado, prioridad, comentarios) se agrupan en bloques visuales con bordes y fondos diferenciados, lo que facilita la lectura y comprensión rápida.

- **Similitud**: Uso de badges con colores y formas consistentes para estados y prioridades. Esto permite al usuario reconocer patrones y asociar rápidamente el significado del color/forma.

- **Figura-Fondo**: El fondo claro y los contenedores con sombra diferencian los tickets y los modales del resto de la interfaz, resaltando la información principal frente al fondo neutro.

- **Continuidad**: La disposición en tabla y tarjetas respeta alineaciones y jerarquías visuales (columnas, títulos, labels), favoreciendo un recorrido visual natural y sin saltos.

### Heurísticas de Nielsen consideradas

1. **Visibilidad del estado del sistema**: El estado del ticket se muestra mediante un badge y se actualiza en tiempo real al cambiarlo, lo que brinda feedback inmediato.

2. **Consistencia y estándares**: Los componentes (select, botones, modales) siguen patrones de interfaz reconocibles y coherentes en toda la aplicación.

3. **Control y libertad del usuario**: El usuario puede cancelar un cambio de estado antes de confirmarlo, evitando errores y dando flexibilidad.

4. **Prevención de errores**: El botón “Guardar” solo aplica el cambio si hay un estado seleccionado, reduciendo errores involuntarios.

5. **Reconocimiento mejor que memorización**: El listado de estados está siempre visible en un Select, evitando que el usuario deba recordar opciones.

6. **Estética y diseño minimalista**: Se evita sobrecargar la pantalla con información; los elementos se presentan en bloques limpios y organizados.

### Contribución a la Experiencia de Usuario

- **Rapidez de comprensión**: La organización visual (Gestalt) permite al usuario identificar de un vistazo estados, prioridades y secciones relevantes.

- **Confianza en el sistema**: Las heurísticas aplicadas (feedback inmediato, consistencia, prevención de errores) generan seguridad al interactuar.

- **Eficiencia**: Filtros y controles claros reducen pasos innecesarios y mejoran la velocidad para encontrar y gestionar tickets.

- **Satisfacción**: Una interfaz limpia, coherente y con respuestas inmediatas hace la interacción más agradable y profesional.

## 4.4. Principios de Accesibilidad y Diseño Inclusivo

Además de los principios de Gestalt y las heurísticas de Nielsen, el diseño de Ticketear incorpora principios fundamentales de **accesibilidad web (WCAG)** para asegurar que la plataforma sea utilizable por el mayor número de personas posible, incluyendo aquellas con discapacidades. El **diseño minimalista** y la **consistencia** del sistema son la base de esta accesibilidad, ya que reducen la carga cognitiva y facilitan la navegación tanto para usuarios nóveles como para aquellos que utilizan tecnologías de asistencia.

Se prioriza una estructura **semántica (HTML5)** para que los lectores de pantalla puedan interpretar correctamente la jerarquía de la información, como los listados de tickets, formularios y modales. Se presta especial atención a la navegación por teclado, asegurando que todos los elementos interactivos (botones, filtros y selectores de estado ) sean completamente funcionales sin depender de un mouse. Finalmente, aunque se utilizan colores para los "badges" de estado, esta información se refuerza con **texto claro** (ej. "Abierto", "En Progreso"), garantizando que el estado sea comprensible sin depender únicamente de la percepción del color.

## 4.5. Prototipo

Prototipo realizado en Figma.

***Link del sitio***: https://icon-glyph-73167849.figma.site

# 5. Casos de Uso (UML)

## 5.1. Diagrama de Casos de Uso

## Caso de Uso (Ficha)

**Caso de uso seleccionado:** *Crear Ticket por Formulario Web (vía API)*

- **Actor principal:** Cliente de la PyME
- **Precondición:** el cliente accede al formulario “Ayuda” en la web de la PyME.
- **Flujo principal:**
    - El cliente completa nombre, apellido, mail y descripción del problema en el formulario.
    - El formulario invoca el endpoint de la API de Ticketear (POST /tickets).
    - El sistema valida los datos recibidos.
    - El sistema crea un ticket en la base de datos.
    - El sistema envía una notificación de confirmación al cliente por email.
    - El ticket queda disponible en el panel del administrador de la PyME.
- **Postcondición:** el ticket queda registrado y con estado inicial “Abierto”.
- **Excepciones:**
    - Si falta un dato obligatorio, el sistema rechaza la creación y devuelve error.

    - Si el servidor de correo no responde, el ticket se crea pero se registra un aviso de fallo en la notificación.

# Diagrama de Casos de Uso (UML)

**Actores principales:**

- Cliente final (externo)
- Administrador de PyME
- Agente de soporte (opcional)
- Sistema externo de Email (SMTP)

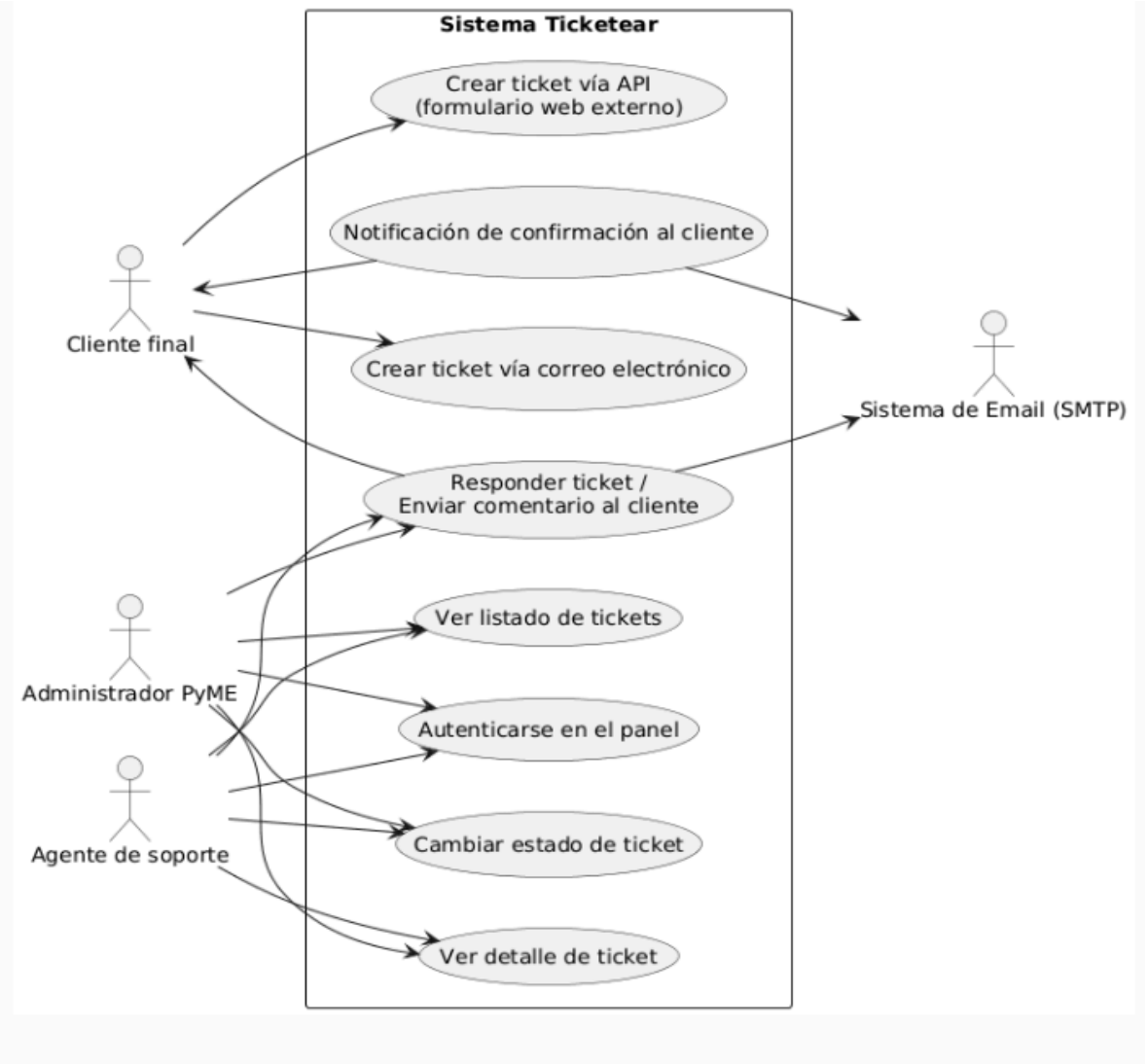
**Casos de uso (MVP):**

1. Crear ticket vía API (formulario web externo)
2. Crear ticket vía correo electrónico
3. Autenticarse en el panel
4. Ver listado de tickets
5. Ver detalle de ticket
6. Cambiar estado de ticket (abierto, en progreso, cerrado)
7. Notificación de confirmación al cliente (cuando se crea el ticket)
8. Responder ticket / Enviar comentario al cliente por emai

## Descripción del Diagrama de Casos de Uso

El Diagrama de Casos de Uso es el punto de partida del modelado y define el alcance funcional del sistema. Este diagrama identifica los actores principales y las interacciones que tendrán con el sistema "Ticketear".

- **Actores:** Se definen claramente tres actores principales:
    - **Cliente Final:** Usuario externo, su única función es **Crear Ticket** (a través de formulario web o email) y **Consultar** el estado de su ticket.
    - **Administrador PyME:** Tiene permisos totales. Es responsable de la **Gestión de Tickets** (ver, modificar, asignar, cerrar) y de gestionar a los Agentes.
    - **Agente de Soporte:** Es un usuario interno. Su rol principal es **Gestionar Tickets** que le fueron asignados, resolverlos, agregar comentarios y cambiar su estado.
- **Propósito:** La principal funcionalidad del sistema es la **Gestión de Tickets**, siendo 'Crear Ticket' y 'Asignar Ticket' las interacciones clave que resuelven el problema de centralización.




# 6. Especificación de Requerimientos

## 6.1. Requerimientos Funcionales:

El sistema debe permitir crear tickets vía API desde formularios externos.

1. El sistema debe procesar correos electrónicos entrantes y convertirlos en tickets.
2. El administrador debe poder loguearse en el panel web y visualizar los tickets creados.
3. El administrador debe poder asignar tickets a agentes internos.

4. El sistema debe enviar notificaciones por email al cliente confirmando la creación del ticket.

## 6.2. Requerimientos no funcionales:

1. **Seguridad:** el sistema debe requerir autenticación por usuario/contraseña segura para acceder al panel administrativo.
2. **Rendimiento:** el sistema debe procesar la creación de tickets en menos de 2 segundos bajo carga normal (≤ 50 tickets por hora).
3. **Usabilidad:** la interfaz del panel debe ser accesible desde navegadores modernos (Chrome, Firefox, Edge) y mostrar listados paginados.

# 7. Recursos Disponibles

## 7.1. Personal (Roles del equipo)

El equipo de "Sistemas Garín S.R.L." está compuesto por 3 integrantes que cubren los roles de análisis, diseño, desarrollo y coordinación del proyecto, tal como lo requiere la guía:

- **Matías Gómez:** Coordinador de Proyecto / Analista Funcional.
- **Leandro Mendoza:** Diseñador UX/UI / Desarrollador Frontend (React).
- **Tomás Patriarca:** Arquitecto de Software / Desarrollador Backend (Node.js).

## 7.2. Materiales (Software y Hardware)

Para la realización del proyecto se cuenta con los siguientes recursos de Software y Hardware :

**Software:**

- **Diseño y Prototipado:** Figma.
- **Desarrollo Backend:** Node.js.
- **Desarrollo Frontend:** React.
- **Base de Datos:** PostgreSQL.
- **Servicios Externos:** Servicio SMTP (ej. SendGrid) para notificaciones por correo.
- **Entorno de Desarrollo (IDEs):** Visual Studio Code.
- **Control de Versiones:** Git / GitHub.

**Hardware:**

- **Desarrollo:** Notebooks y PCs personales de los integrantes del equipo.
- **Despliegue (Producción):** Servidores en la Nube (Cloud) para el alojamiento de la API y la Base de Datos.

# 8. Plan de Trabajo

El plan de trabajo del proyecto "Ticketear" se divide en dos grandes fases, alineadas con los dos cuatrimestres de la Práctica Profesionalizante

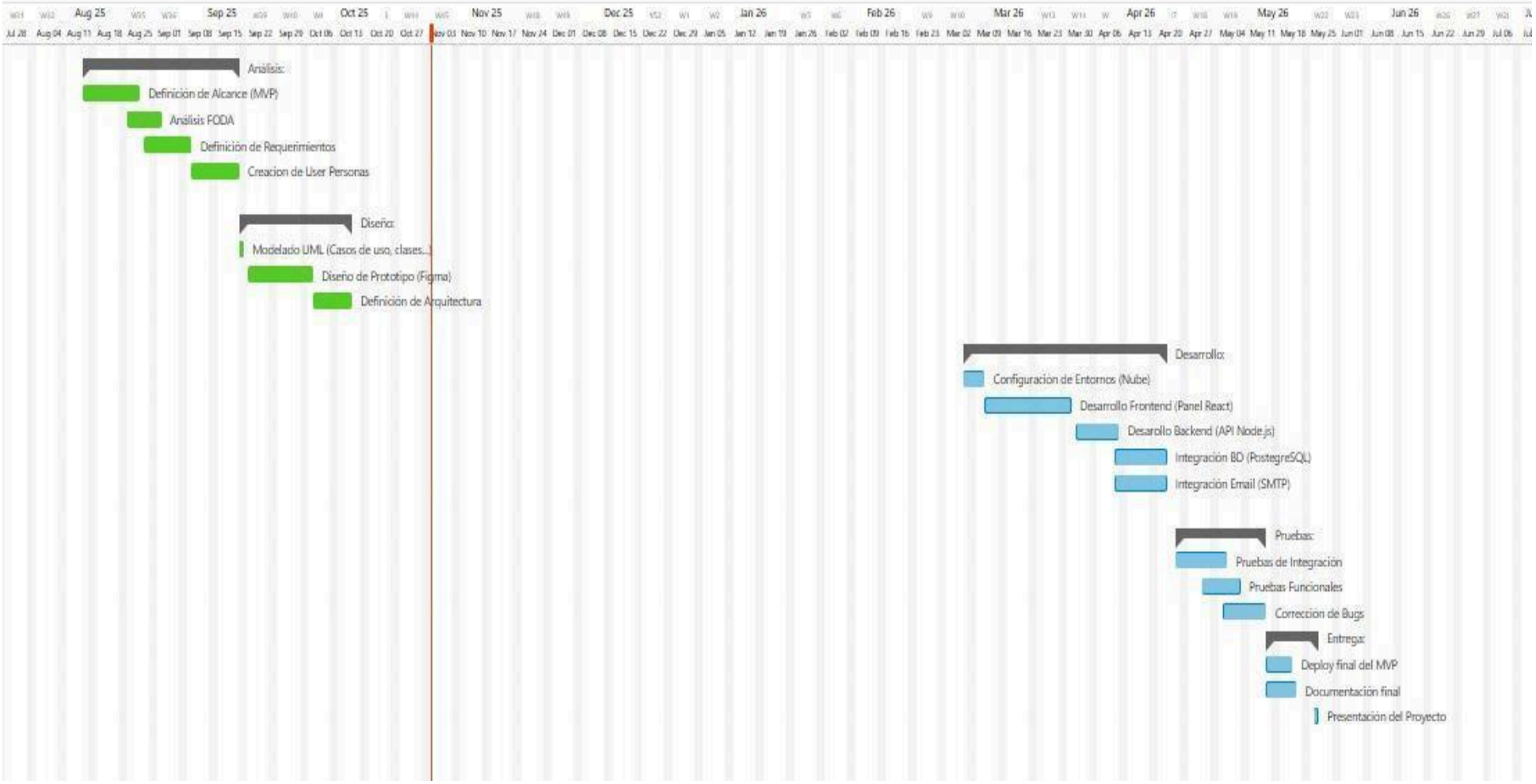
- **Fase 1 (Agosto - Noviembre):** Centrada en el Análisis y Diseño del sistema. Incluye la definición de requerimientos, el modelado UML y la creación del prototipo navegable.
- **Fase 2 (Marzo - Junio):** Centrada en el Desarrollo, Pruebas y Entrega. Incluye la construcción del software (Backend y Frontend), la integración, las pruebas funcionales y el despliegue final.

## 8.1 Hitos del Proyecto

Los hitos principales que marcan el progreso del proyecto son:

1. **Análisis:** Definición completa del alcance, FODA y requerimientos.
2. **Diseño:** Entrega del prototipo navegable (Figma) y la arquitectura del sistema.
3. **Desarrollo:** Integración funcional del Backend (API) y Frontend (Panel).
4. **Pruebas:** Finalización de la corrección de *bugs* críticos y validación funcional (QA).
5. **Entrega:** Despliegue del MVP en un servidor y presentación final

## 8.2 Cronograma



## Resumen del Cronograma:

- **Hito 1: Análisis (Agosto - Octubre)**
    - Definición de Alcance (MVP): ~3 semanas (Fin de Ago - Principio de Sep)
    - Análisis FODA: ~2 semanas (Principio de Sep - Mediados de Sep)
    - Definición de Requerimientos: ~3 semanas (Mediados de Sep - Fin de Sep)
    - Creación de User Personas: ~2 semanas (Fin de Sep - Principio de Oct)
- **Hito 2: Diseño (Octubre - Noviembre)**
    - Modelado UML: ~3 semanas (Principio de Oct - Mediados de Oct)
    - Diseño de Prototipo (Figma): ~3 semanas (Mediados de Oct - Fin de Oct)
    - Definición de Arquitectura: ~2 semanas (Fin de Oct - Principio de Nov)
- **Hito 3: Desarrollo (Marzo - Mayo)**
    - Configuración de Entornos (Nube): ~2 semanas (Principio de Mar - Mediados de Mar)
    - Desarrollo Frontend (Panel React): ~4 semanas (Mediados de Mar - Principio de Abr)
    - Desarrollo Backend (API Node.js): ~4 semanas (Fin de Mar - Mediados de Abr)
    - Integración BD (PostgreSQL): ~3 semanas (Principio de Abr - Fin de Abr)
    - Integración Email (SMTP): ~3 semanas (Mediados de Abr - Principio de May)
- **Hito 4: Pruebas (Mayo - Junio)**
    - Pruebas de Integración: ~2 semanas (Principio de May - Mediados de May)
    - Pruebas Funcionales: ~2 semanas (Mediados de May - Fin de May)
    - Corrección de Bugs: ~2 semanas (Fin de May - Principio de Jun)
- **Hito 5: Entrega (Junio)**
    - Deploy final del MVP: ~1 semana (Principio de Jun)
    - Documentación final: ~1 semana (Principio de Jun)
    - Presentación del Proyecto: ~1 semana (Mediados de Jun

# 9. Gestión de Riesgos

| Tipo de Riesgo | Riesgo Identificado | Propuesta de Mitigación |
|---|---|---|
| **Disponibilidad** | **Fallas de servidor (Caídas):** El servidor de aplicación o base de datos deja de funcionar, impidiendo el acceso al panel y la creación de tickets. | Desplegar la aplicación en servicios de nube (Cloud) con alta disponibilidad (ej. AWS, Azure, Vercel) que garanticen un 99.9% de *uptime* y ofrezcan *backups* automáticos de la base de datos. |
| **Seguridad** | **Seguridad de datos (Brecha):** "Riesgo de problemas de seguridad si no se configuran bien los accesos por API. | Implementar autenticación robusta (JWT) para el panel de admin. Forzar HTTPS en todas las comunicaciones. Asegurar la API con CORS y validar todos los datos de entrada para prevenir inyección SQL. |
| **Confiabilidad** | **Dependencia de servicios externos:** "Dependencia de servicios externos para correo... (ej. SMTP, SendGrid)". Una falla en este servicio impediría las notificaciones. | Seleccionar un proveedor SMTP robusto y con buena reputación. Implementar un sistema de *logs* (registros) que alerte si los correos fallan, y un mecanismo de reintentos automáticos para el envío de notificaciones críticas. |

| | | |
|---|---|---|
| **Usabilidad** | **Resistencia a la adopción:** "Posible resistencia de las PyMEs a integrar sus formularios con APIs externas". | Ofrecer la **creación de tickets por email** como la alternativa principal, ya que no requiere integración técnica. Para la API, proveer documentación muy clara, simple y con ejemplos de "copiar y pegar". |

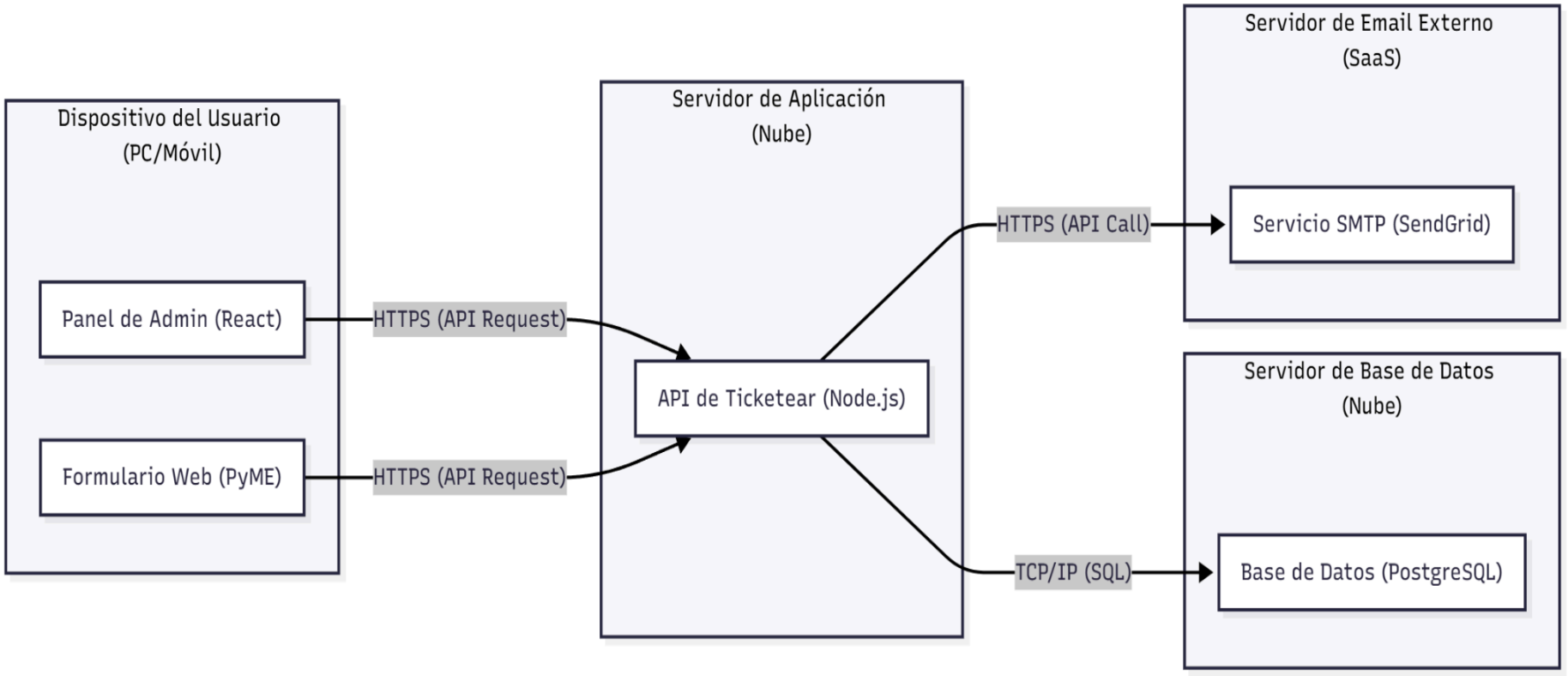
# 10. Diseño del Sistema

## 10.1. Arquitectura Cliente-Servidor

### Descripción del Diagrama de Despliegue

El Diagrama de Despliegue representa la arquitectura **Cliente-Servidor (tres capas)** del sistema y cómo se distribuyen los componentes de software en el hardware físico.

- **Cliente (Frontend):** Representado por un navegador web, ejecuta la interfaz de usuario desarrollada en **React**. Su única función es comunicarse con el Servidor mediante llamadas a la API.
- **Servidor (Backend):** Nodo central donde reside la lógica. Contiene el componente **API RESTful** (desarrollada en Node.js) que se encarga de procesar las solicitudes, aplicar la lógica de negocio y comunicarse con la Base de Datos.
- **Base de Datos (PostgreSQL):** Componente de persistencia donde se almacenan todos los datos críticos (Tickets, Usuarios, Comentarios). La API es el único componente autorizado para leer y escribir en este nodo, asegurando la integridad de los datos.
- **Servicios Externos:** Incluye el servicio **SMTP** (ej. SendGrid) que es utilizado por el Servidor de Notificaciones para enviar los correos de confirmación a los clientes.


Dispositivo del Usuario
(PC/Móvil)
Panel de Admin (React)
Formulario Web (PyME)
HTTPS (API Request)
HTTPS (API Request)
Servidor de Aplicación
(Nube)
API de Ticketear (Node.js)
HTTPS (API Call)
TCP/IP (SQL)
Servidor de Email Externo
(SaaS)
Servicio SMTP (SendGrid)
Servidor de Base de Datos
(Nube)
Base de Datos (PostgreSQL)

## 10.2. Diagrama de actividades

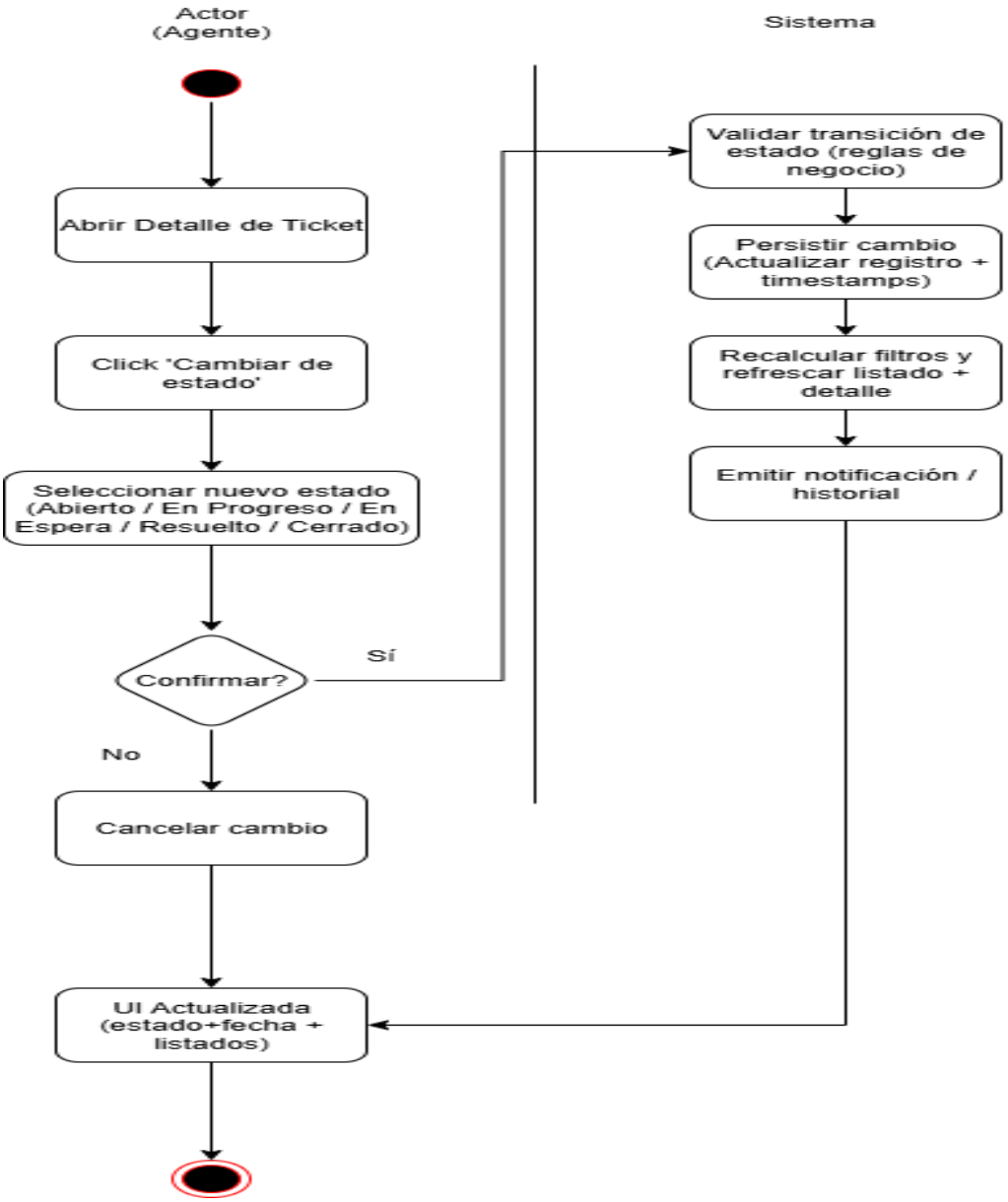
Aquí tienes la descripción completa y final del **Diagrama de Actividades** para tu documento, lista para ser insertada después del título de la sección:

### Descripción del Diagrama de Actividades

El Diagrama de Actividades se utiliza para modelar el flujo de trabajo del proceso clave de **"Cambiar estado de ticket"**, ilustrando las acciones y transiciones que ocurren en el sistema.

El diagrama está dividido en dos carriles (*Swimlanes*) para separar claramente las responsabilidades: lo que hace el **Actor (Agente)** y lo que realiza el **Sistema**.

- **Ruta del Agente:** El flujo comienza cuando el Agente inicia la acción (Abrir Detalle de Ticket y Click 'Cambiar de estado'). Después de Seleccionar nuevo estado, el Agente llega a un punto de decisión (Confirmar?). Si elige No, el proceso se cancela.
- **Ruta del Sistema:** Si el Agente elige `Sí`, el Sistema toma el control para asegurar la coherencia:
    1. Realiza una Validar transición de estado (reglas de negocio).
    2. Persistir cambio (actualizando el registro y timestamps en la base de datos).
    3. Recalcular filtros y refrescar listado + detalle.
    4. Emitir notificación / historial.
- **Propósito:** Este modelado demuestra que el proceso de gestión de tickets no es solo una acción secuencial, sino que incorpora **reglas de negocio** y garantiza que el registro del ticket se actualice y que el cliente final sea notificado de los cambios.

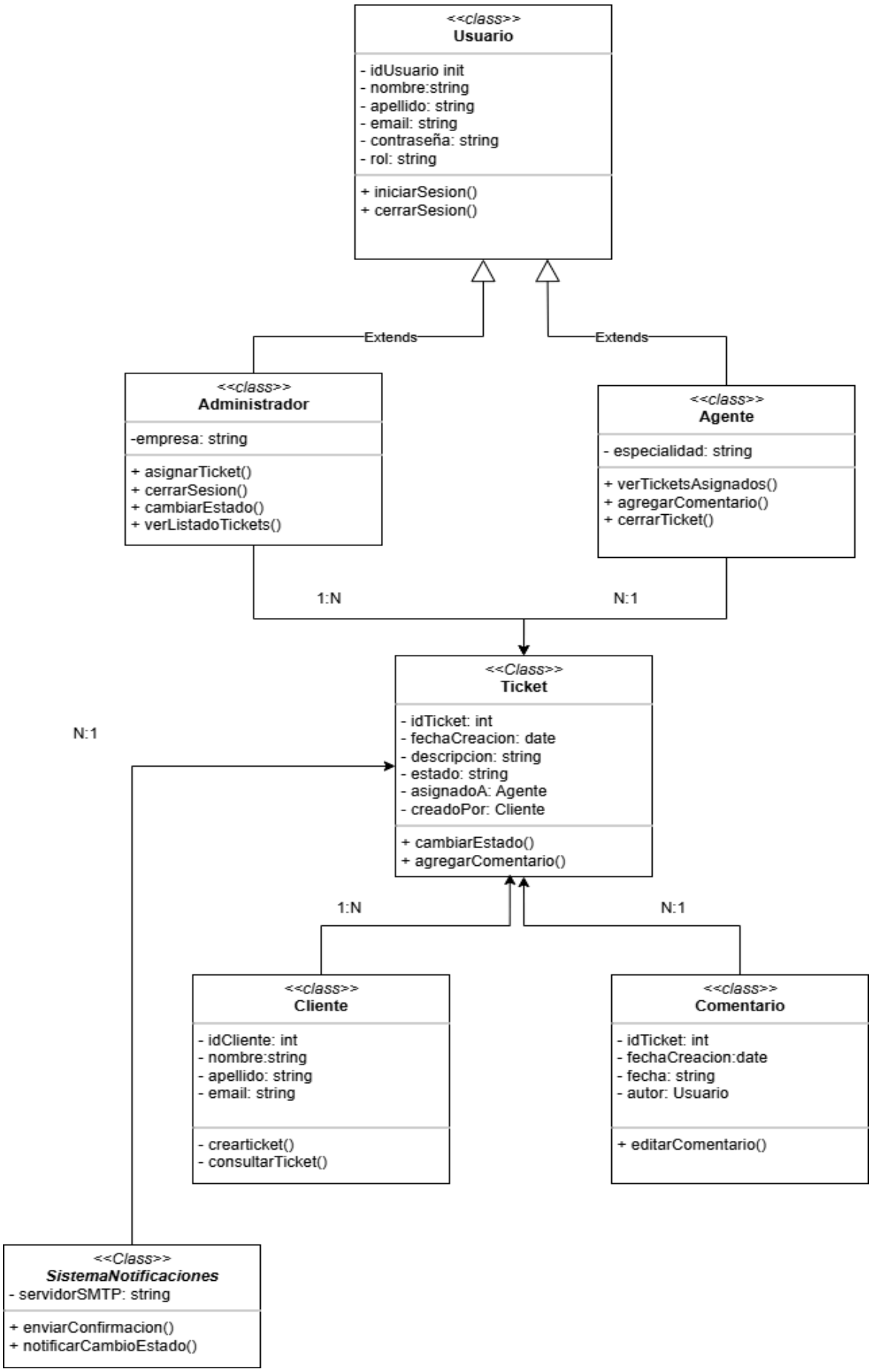

Actor
(Agente)
Sistema
Abrir Detalle de Ticket
Click 'Cambiar de estado'
Seleccionar nuevo estado (Abierto / En Progreso / En Espera / Resuelto / Cerrado)
Confirmar?
Sí
No
Cancelar cambio
UI Actualizada (estado+fecha + listados)
Validar transición de estado (reglas de negocio)
Persistir cambio (Actualizar registro + timestamps)
Recalcular filtros y refrescar listado + detalle
Emitir notificación / historial

## 10.3. Diagrama de Clases

### Descripción del Diagrama de Clases

El Diagrama de Clases es el plano estructural del sistema. Modela las entidades de negocio y las relaciones estáticas entre ellas, sirviendo de base para el diseño de la base de datos.

- **Clases Clave:** Las clases principales son `Ticket`, `Administrador`, `Agente` y `Cliente`.
- **Relaciones:** Se define una relación de **Composición** entre `Ticket` y `Comentario` (los comentarios no pueden existir sin un ticket). Las relaciones de **Asociación (1 a N)** definen que un `Cliente` crea muchos `Ticket`, y un `Administrador` gestiona muchos `Ticket`.
- **Herencia:** Las clases `Administrador` y `Agente` heredan atributos y métodos de la clase base `Usuario`, aplicando el concepto de herencia para definir los roles.
- **Propósito:** Este diagrama garantiza que las reglas de negocio, como la asignación de responsables y la creación de tickets, se reflejen correctamente en la estructura de la información.

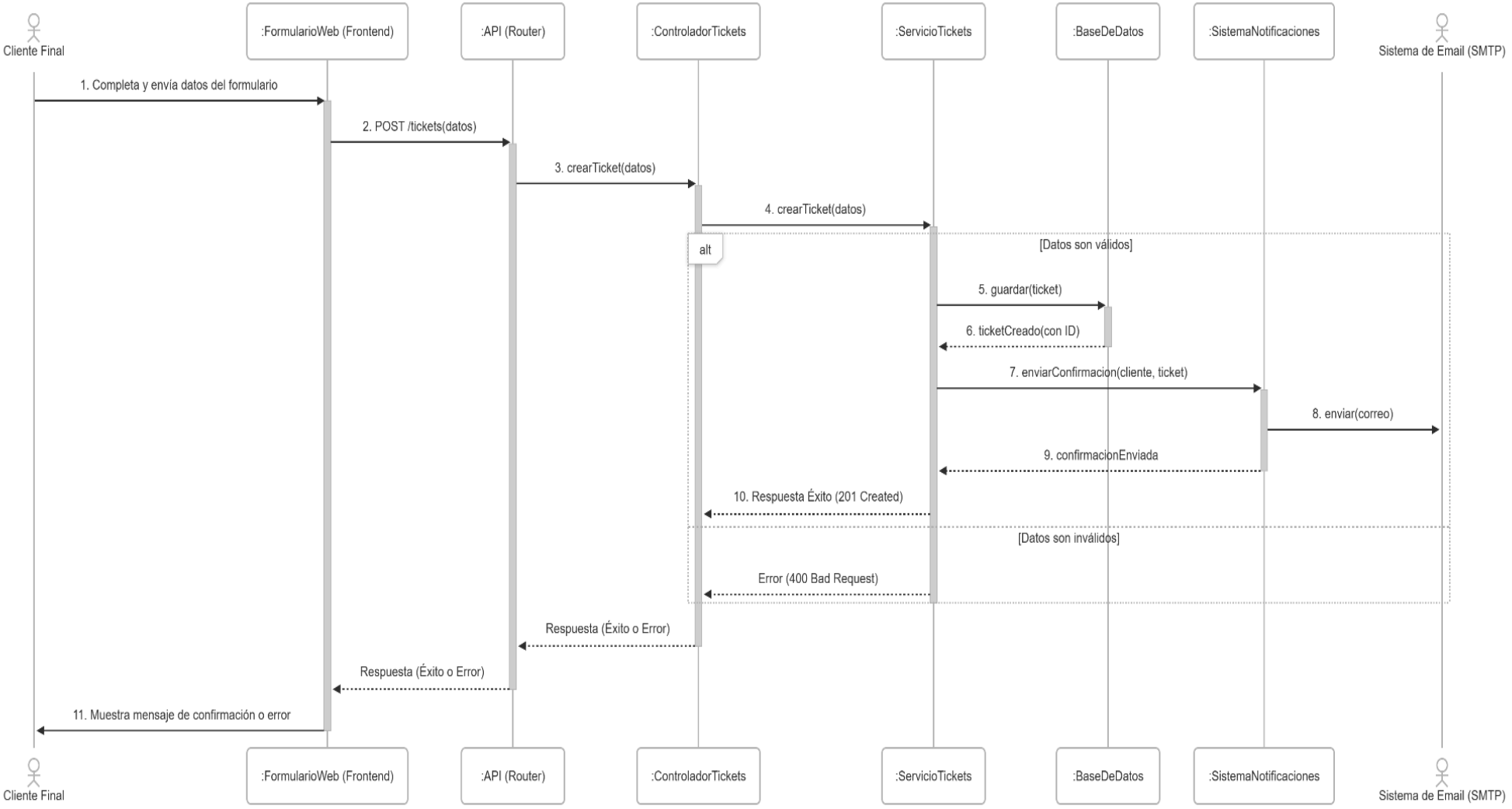

<<class>>
Usuario
- idUsuario init
- nombre:string
- apellido: string
- email: string
- contraseña: string
- rol: string
+ iniciarSesion()
+ cerrarSesion()
Extends
Extends
<<class>>
Administrador
-empresa: string
+ asignarTicket()
+ cerrarSesion()
+ cambiarEstado()
+ verListadoTickets()
<<class>>
Agente
- especialidad: string
+ verTicketsAsignados()
+ agregarComentario()
+ cerrarTicket()
1:N
N:1
<<Class>>
Ticket
- idTicket: int
- fechaCreacion: date
- descripcion: string
- estado: string
- asignadoA: Agente
- creadoPor: Cliente
+ cambiarEstado()
+ agregarComentario()
N:1
1:N
N:1
<<class>>
Cliente
- idCliente: int
- nombre:string
- apellido: string
- email: string
- crearticket()
- consultarTicket()
<<class>>
Comentario
- idTicket: int
- fechaCreacion:date
- fecha: string
- autor: Usuario
+ editarComentario()
<<Class>>
SistemaNotificaciones
- servidorSMTP: string
+ enviarConfirmacion()
+ notificarCambioEstado()

## 10.4. Diagrama de Secuencia

### Descripción del Diagrama de Secuencia

El diagrama muestra el flujo de interacciones que ocurren desde que el cliente envía el formulario hasta que el ticket es creado y se le notifica.

1. **Inicio**: El Cliente Final completa los datos en el :FormularioWeb y lo envía.
2. Petición API: El formulario realiza una llamada POST /tickets a la :API de Ticketear, enviando los datos del cliente y el problema.
3. **Control de la Petición**: La :API redirige la petición al :ControladorTickets, que invoca al método crearTicket().
4. **Lógica de Negocio**: El :ControladorTickets pasa los datos al :ServicioTickets. Este servicio primero valida que todos los campos obligatorios estén presentes. Si un dato falta, se devuelve un mensaje de error que se propaga hasta el cliente.
5. **Persistencia**: Si los datos son válidos, el :ServicioTickets crea una nueva instancia de Ticket y solicita a la :BaseDeDatos que guarde el registro. La base de datos confirma la operación y devuelve el ticket con su nuevo ID.
6. **Notificación**: Con el ticket ya creado, el :ServicioTickets invoca al :SistemaNotificaciones para enviar un correo de confirmación al cliente.
7. **Envío de Email**: El :SistemaNotificaciones se comunica con el actor externo :Sistema de Email (SMTP) para despachar el correo electrónico.
8. **Respuesta Final**: Una vez completados los pasos, se devuelve una respuesta de éxito (HTTP 201 Created) a través de todas las capas hasta llegar al :FormularioWeb, confirmando al cliente que su ticket fue registrado. El ticket queda en estado "Abierto" y disponible en el panel del administrador.


Cliente Final
:FormularioWeb (Frontend)
:API (Router)
:ControladorTickets
:ServicioTickets
:BaseDeDatos
:SistemaNotificaciones
Sistema de Email (SMTP)
1. Completa y envía datos del formulario
2. POST /tickets(datos)
3. crearTicket(datos)
4. crearTicket(datos)
alt
[Datos son válidos]
5. guardar(ticket)
6. ticketCreado(con ID)
7. enviarConfirmacion(cliente, ticket)
8. enviar(correo)
9. confirmacionEnviada
10. Respuesta Éxito (201 Created)
[Datos son inválidos]
Error (400 Bad Request)
Respuesta (Éxito o Error)
Respuesta (Éxito o Error)
11. Muestra mensaje de confirmación o error

# 12. Conclusiones Finales

## 12.1. Reflexión del Equipo sobre el Aprendizaje Logrado

La realización de "Ticketear" ha sido un ejercicio integral que nos permitió conectar la teoría del diseño de sistemas con un caso de práctica profesional. El mayor aprendizaje fue el proceso de traducir una necesidad de negocio concreta (el "caos" de tickets en las PyMEs) en un conjunto completo de artefactos de software: desde la definición de los actores y sus necesidades (User Personas) , pasando por el modelado formal del sistema (diagramas UML de Casos de Uso, Clases, Actividad y Secuencia) , hasta la definición de una arquitectura cliente-servidor (React, Node.js) lista para ser implementada.

Este proyecto nos consolidó como equipo en la toma de decisiones de arquitectura y en cómo priorizar funcionalidades (definición de MVP) para entregar valor real a un cliente.

## 12.2. Evaluación y Mejoras Futuras del Sistema

La principal fortaleza de "Ticketear" es su simplicidad y su enfoque directo en el problema de la PyME , ofreciendo canales de creación de tickets claros (API y Email) sin la sobrecarga de funciones de competidores más grandes. Su principal debilidad, identificada en el FODA, es la dependencia de servicios externos de SMTP (como SendGrid) y la falta de reportes avanzados en esta primera versión.

Como mejoras futuras, el próximo paso natural del proyecto sería implementar las funcionalidades que quedaron estratégicamente fuera del alcance de este MVP:

- Integración con canales de mensajería instantánea, siendo WhatsApp el más prioritario para el mercado local.
- Un módulo de reportes complejos y analítica avanzada para que el Administrador pueda medir tiempos de respuesta (SLA) y el rendimiento de sus agentes.
- Una base de conocimiento o portal de autoayuda para que los clientes finales puedan resolver dudas comunes sin necesidad de crear un ticket.