UNIVERSIDAD ADOLFO IBÁÑEZ
ARQUITECTURA DE SISTEMAS
PROFESOR JUAN GANA
SEMESTRE I 2025

Gestión de encomiendas en edificios

GRUPO E

María Rosario Baeza

Felipe Fesser

Vicente Garcia

Marín Osorio

Renato Valdés

1. **Evaluación inicial PoC**

Escenario	**Descripción**	**Métrica a Evaluar**	**Herramientas Utilizadas**	**Resultado Esperado**	**Posibles Cuellos de Botella**	**Recomendación**
Carga de Registro de Paquetes	Simulación del registro simultáneo de 100 paquetes por parte de los conserjes.	Tiempo de respuesta de la API, uso de base de datos, consumo de recursos.	JMeter, Locust, Datadog	Tiempo de respuesta <1 segundo, uso moderado de recursos.	Problemas con las operaciones concurrentes de MongoDB.	Optimización de MongoDB, implementación de índices.
Alta Frecuencia de Notificaciones	Simulación del envío simultáneo de 1000 notificaciones a los residentes sobre el estado de sus paquetes.	Tiempo de entrega de correos, rendimiento de la API, uso de recursos.	Locust, Datadog	Tiempo de envío < 5 segundos para 1000 notificaciones, alto uso de CPU y red.	Sobrecarga del sistema de correos.	Implementar un sistema de notificaciones más robusto.
Simulación de Usuarios Concurrentes	Simulación de 65 usuarios concurrentes realizando varias acciones: residentes, conserjes y administradores.	Tiempo de respuesta de la API, uso de infraestructura (CPU, memoria, ancho de banda).	JMeter, Locust, Datadog	Tiempo de respuesta <2 segundos por acción, uso moderado de recursos.	Problemas de escalabilidad sin soporte horizontal.	Considerar escalabilidad horizontal y sharding en MongoDB.

2. **Propuesta de arquitectura to-be**

Arquitectura de Procesos simples

Nuestra propuesta de Arquitectura es la siguiente:

- **Recepción de la encomienda**: Un conserje recibe un paquete y lo registra en el sistema. Donde el paquete se asigna a un residente específico.
- **Notificación**: Una vez que el paquete está registrado, se envía una notificación automática al residente con los detalles de la entrega.
- **Retiro del paquete**: El residente accede al sistema, visualiza el estado del paquete, y valida el retiro mediante un código QR o código único.
- **Historial de entregas**: El sistema mantiene un historial de todos los paquetes entregados, permitiendo que los residentes y conserjes consulten el estado de los envíos pasados.

Mejoras a implementar:

- Mejorar la integración entre el módulo de notificación y la gestión de alertas (para integrar SMS y notificaciones push).
- Automatizar y mejorar la verificación de paquetes con sistemas de autenticación robusta (como JWT).

A continuación se presenta en diagrama de Procesos Simple:

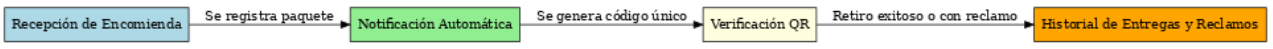

Figura 1: Diagrama de procesos simple

Arquitectura de Aplicaciones y Datos

Este diagrama muestra la relación entre los diferentes módulos del sistema, el flujo de datos y cómo interactúan las aplicaciones y la base de datos. Aquí se detallan los cambios propuestos:

- **Frontend**: La interfaz de usuario se mantendrá en React + TypeScript, pero con mejoras en la experiencia móvil y optimización de la interfaz para dispositivos móviles.
- **Backend**: Se mantendrá el sistema Deno + Oak como servidor, pero se incluirán mejoras de escalabilidad y seguridad con JWT y autenticación basada en roles.
- **Base de Datos**: Se actualizará el uso de MongoDB para garantizar la utilización de índices eficientes y un sistema de replicación para mejorar el rendimiento y la disponibilidad.
- **Notificaciones**: Se integrarán notificaciones push junto con las ya existentes por correo electrónico, y se explorará la integración con SMS.

Mejoras a implementar:

- **Escalabilidad**: Implementar un sharding en MongoDB para manejar un mayor volumen de usuarios y datos, especialmente si se da la oportunidad de escalar el sistema a otros edificios.
- **Seguridad**: Integración de un sistema de autenticación más robusto utilizando OAuth 2.0 o JWT.
- **Optimización**: Mejorar el tiempo de respuesta de las APIs y reducir la carga en el backend con microservicios si es necesario.

A continuación se presenta en diagrama de Aplicaciones y datos:

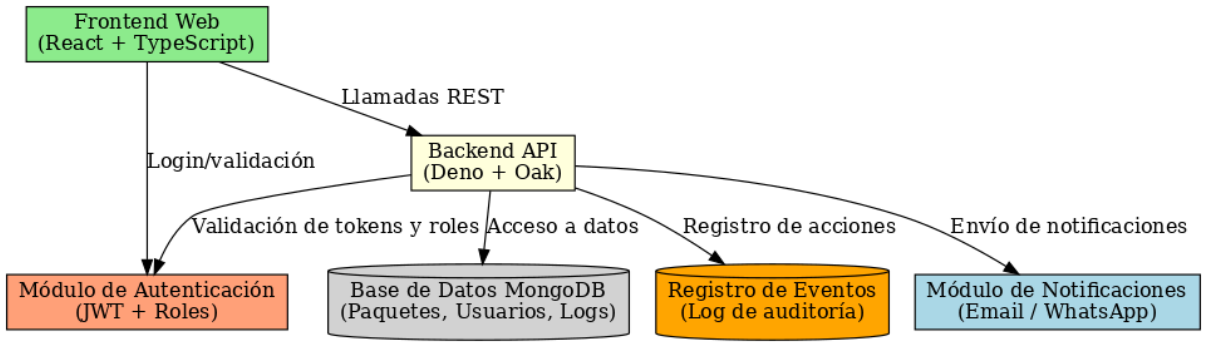

Figura 2: Diagrama de aplicaciones y datos

Arquitectura de Infraestructura

En cuanto a la infraestructura, la propuesta se enfoca en mejorar la disponibilidad, la escalabilidad y la seguridad.

- **Infraestructura en la nube:** Para asegurar alta disponibilidad, se propone desplegar el sistema en un proveedor de nube pública como AWS o Azure, habilitando autoscaling para adaptarse dinámicamente a la carga.
- **Balanceador de carga:** Se incorporará un balanceador de carga para distribuir las solicitudes entrantes entre múltiples instancias del backend, permitiendo una arquitectura escalable horizontalmente.
- **CDN (Content Delivery Network):** Para optimizar la entrega del frontend (React), se utilizará un CDN que pueda almacenar en caché los archivos estáticos y reduzca la latencia para los usuarios.
- **Base de datos replicada:** Se utilizará MongoDB Atlas con replicación geográfica para así asegurar la alta disponibilidad y recuperación ante fallos.
- **Servicio de notificaciones externo:** Se integrará un servicio de terceros como SendGrid o Twilio para manejar el envío de correos electrónicos y mensajes instantáneos de forma desacoplada.

- **Servicio de monitoreo:** Se incluirá un sistema de monitoreo (como Datadog o Prometheus) para registrar métricas de uso, generar alertas y anticipar problemas.

- **Seguridad:** Se reforzará la infraestructura con firewalls, VPNs, control de acceso y monitoreo continuo para prevenir que no hayan accesos no autorizados y lograr garantizar la integridad del sistema.

A continuación se presenta en diagrama de Infraestructura:

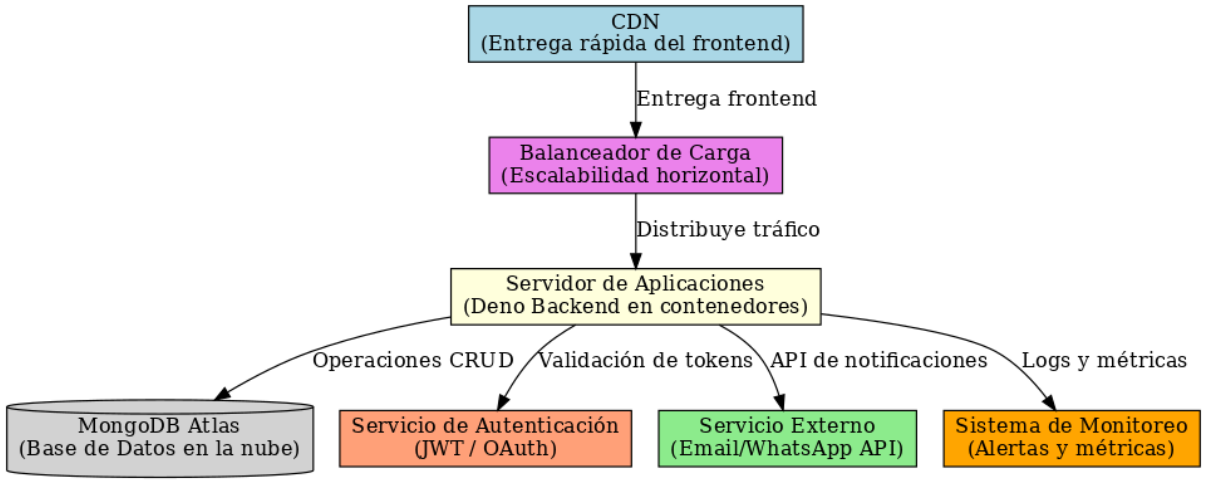

Figura 3: Diagrama de Infraestructura

3. Justificación de decisiones arquitectónicas

La arquitectura propuesta del sistema de gestión de encomiendas responde a las brechas detectadas y a los objetivos estratégicos del cliente, alineándose con los principios de escalabilidad, trazabilidad, seguridad y disponibilidad. Se tomaron seis grandes decisiones con respecto a la arquitectura.

En primer lugar está la incorporación de una autenticación robusta con JWT y control de roles, buscando minimizar el riesgo que presentaba los accesos poco seguros, sin diferenciación clara entre usuarios, lo que podría generar que se vulnerara el acceso de algún usuario. Esto permite autenticar y autorizar usuarios con distintos privilegios, como residente, conserje y administrador, mejorando la seguridad y reduciendo el riesgo de accesos indebidos a información sensible.

La segunda decisión tomada fue la adición de un módulo de notificaciones desacoplado, como email y WhatsApp, buscando diferenciarse de las notificaciones poco confiables o enviadas de forma manual, como lo son los SMS, donde se tiende a desconfiar de su procedencia e intención. Esto permite escalar la cantidad de notificaciones sin sobrecargar el backend. Así también mejorando el rendimiento y la experiencia de usuario al garantizar que los residentes reciban alertas inmediatas y automáticas a la llegada de sus paquetes.

Como tercera decisión está la implementación de auditoría y logs de eventos, esto surge a raíz de la falta de trazabilidad en acciones críticas como retiro de encomiendas o reclamos. El registro de eventos permite reconstruir el historial de actividades y es clave para resolver conflictos, respaldando la confiabilidad y transparencia del sistema.

La cuarta decisión tomada fue el uso de MongoDB Atlas y escalabilidad horizontal como forma de resolver problemas de rendimiento en operaciones concurrentes y riesgos ante aumentos de carga. Al escalar horizontalmente y usar un servicio de base de datos gestionado, se garantiza mayor disponibilidad y escalabilidad, preparando al sistema para ser usado en múltiples edificios o contextos.

Otra decisión tomada fue la integración con sistemas de monitoreo para resolver la falta de visibilidad sobre el estado del sistema. Esto facilita el diagnóstico de fallos y el mantenimiento preventivo, apoyando la mantenibilidad del sistema y reduciendo el tiempo de inactividad.

Por último está la incorporación de CDN y balanceador de carga para resolver la latencia en la carga del frontend y riesgo de saturación del backend. Esto optimiza la entrega de archivos estáticos, mejora la experiencia de usuario y permite manejar más usuarios simultáneos mediante la distribución de tráfico entre servidores backend.

Cada decisión arquitectónica está directamente relacionada con una brecha técnica y un atributo de calidad priorizado. La arquitectura to-be no solo mejora el funcionamiento actual del sistema, sino que lo posiciona para escalar, integrarse con nuevas tecnologías y brindar una experiencia más robusta y segura a sus usuarios finales.

4. **Mejoras a nivel de código y patrones**

Implementación del patrón Controller-Service-Repository

En el backend (Deno + Oak), gran parte de la lógica de negocio se encuentra mezclada directamente dentro de las rutas, lo que genera dificultad para escalar o testear.
Nuestra propuesta consiste en aplicar el patrón **Controller-Service-Repository**, dividiendo claramente las responsabilidades:

Controller	Maneja las rutas y responde a solicitudes HTTP.
Service	Contiene la lógica de negocio (validaciones, reglas, etc.).
Repository	Abstracción del acceso a la base de datos (MongoDB).

El impacto de estas divisiones serian:

- Mejora la **mantenibilidad**
- Facilita la implementación de **tests unitarios**
- Favorece la reutilización y desacoplamiento del código
- Permite evolucionar la lógica sin modificar el controlador

5. **Discusión y conclusión**

El rediseño arquitectónico qué se propone mejora significativamente la calidad del sistema, alineándolo de manera efectiva con los objetivos del cliente y las exigencias operativas de edificios residenciales qué requieren un funcionamiento continuo, 24/7. Las decisiones claves, cómo la adopción del Controller-Service-Repository, la integración de un sistema de autenticación robusto basado en JWT, y la optimización del sistema de base de datos con MongoDB Atlas, han resultado en mejoras vitales en la escalabilidad, seguridad y longevidad del sistema.

La adopción de esté enfoque arquitectónico proporciona una infraestructura más flexible, capaz de manejar cargas de trabajo más altas y de adaptarse a futuros desafíos, cómo una

eventual expansión a varios edificios o el lanzamiento de una interfaz móvil. Está arquitectura está diseñada no solo para satisfacer las necesidades actuales, sino las futuras, permitiendo una evolución continua sin comprometer integridad o rendimiento del sistema.

A lo largo del proceso, se destacó que la calidad de un sistema no se limita a su funcionalidad, sino qué también depende de su capacidad para mantenerse y evolucionar sin fallas en un entorno real. La experiencia qué adquirimos en esté proyecto nos permitió fortalecer nuestro conocimiento técnico y comprender profundamente el valor de alinear la tecnología con los objetivos del negocio y las necesidades de los usuarios. El resultado es una solución digital sostenible, diseñada para escalar y adaptarse a las demandas del mercado.

En conclusión, el nuevo diseño arquitectónico resuelve las limitaciones técnicas de la versión actual y posiciona al sistema cómo una solución robusta y preparada para enfrentar exigencias de un entorno dinámico y de alto rendimiento. La capacidad para integrar nuevas funcionalidades y mejorar la experiencia del usuario final garantiza qué esté sistema no solo sea eficiente, sino que sea confiable y listo para cualquier eventualidad en el futuro.