

RFI II. Proyecto Votación - Arquitectura Basada en Microservicios

Autores: Oier A., Urki A., Oier L., Javier P., Álex S.

Metodología y buenas prácticas

METODOLOGÍA DE TRABAJO

Tarea Terminada

- Implementadas especificaciones de issue
- Revisada por linting y pruebas autom.
- Documentación actualizada

Flujo Código

1. Implementar issue en local
2. Commit en rama desarrollo
3. PR para mergear rama desarrollo en main
4. Comentarios en PR con corrección de código
5. Pruebas CI con corrección de código





Aprobación Cambios

- PR revisado por al menos otro desarrollador

Documentación

- Swagger para documentar APIs

Despliegue + DevOps

- CI/CD →  GitHub Actions
- Monitoreo + Logging →  Kong
- Contenedores →  docker
- Control de versiones →  GitHub

Metodología y buenas prácticas

BUENAS PRÁCTICAS EN EL BACKEND

Linting

StyleCop: 

- ✔ Reglas de Espaciado (SA1000–): Aseguran el uso adecuado de espacios alrededor de palabras clave y símbolos en el código.
- ✔ Reglas de Legibilidad (SA1100–): Garantizan que el código esté bien formateado y sea fácil de leer.
- ✔ Reglas de Orden (SA1200–): Establecen un orden estándar para los contenidos del código.
- ✔ Reglas de Nomenclatura (SA1300–): Enfocadas en las convenciones de nombres para miembros, tipos y variables.
- ✔ Reglas de Diseño (SA1500–): Enfocadas en el diseño y el interlineado en los archivos de código fuente.

Testing

DotNet:



Dado que estamos desarrollando en C#, un lenguaje estrechamente relacionado con .NET, hemos decidido utilizar directamente DotNet para ejecutar los tests implementados.
Dentro del proceso de integración continua (CI), se ha añadido una fase de pruebas en la que se ejecutan todos los tests tras cada commit.

Metodología y buenas prácticas

BUENAS PRÁCTICAS EN EL FRONTEND

Linting

ASTRO: 

- ✔ - **no-set-html-directive** → Evita directivas HTML
- ✔ - **no-unused-css-selector** → Detecta selectores CSS sin usar
- ✗ - **no-inline-styles** → No permite el uso de estilos en línea.
- ✗ - **no-unused-components** → Detecta componentes Astro sin usar

TypeScript: 

- ✔ - **no-unused-vars** → Detecta variables TS no usadas
- ✔ - **no-var-requires**
- ✔ - **no-require-imports**
- ✗ - **explicit-function-return-type** → Exige que las funciones tengan tipo de retorno explícito

MICROSERVICIOS y FUNCIONALIDAD

Candidatos



- Gestión de la información relacionada con los candidatos
 - * Información personal
 - * Votos
- Desarrollo de Web API (CRUD)

GET /api/Candidates/ObtenerCandidatoPorId/{id}

POST /api/Candidates/InsertarNuevoCandidato

PUT /api/Candidates/ActualizarVotosCandidato

DELETE /api/Candidates/EliminarCandidato/{id}

Autenticación y autorización



- JWT: forma segura de compartir información
- Necesario para consultar la API Candidatos
- Genera token con 1 hora de validez
- Privilegios de administrador y normal
- Contraseñas - Base64 - hashear SHA-512

Authorization

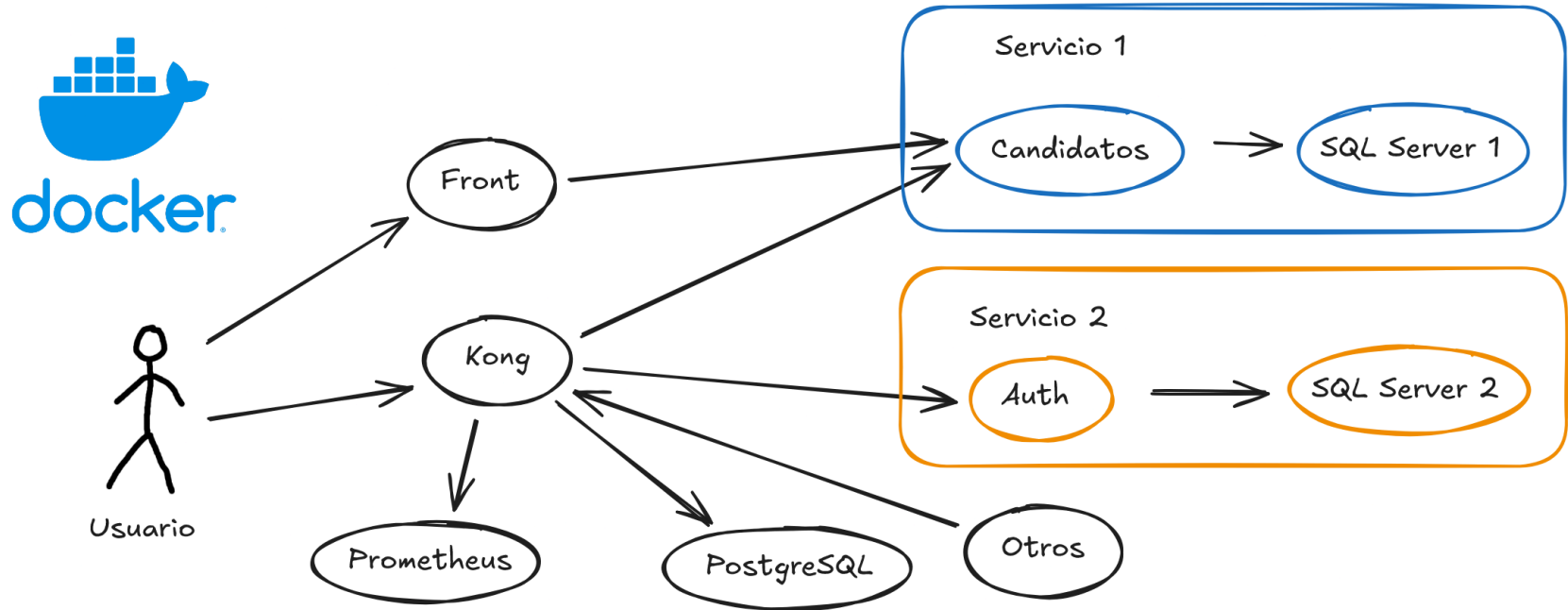
POST /api/Auth/Login

POST /api/Auth/Register

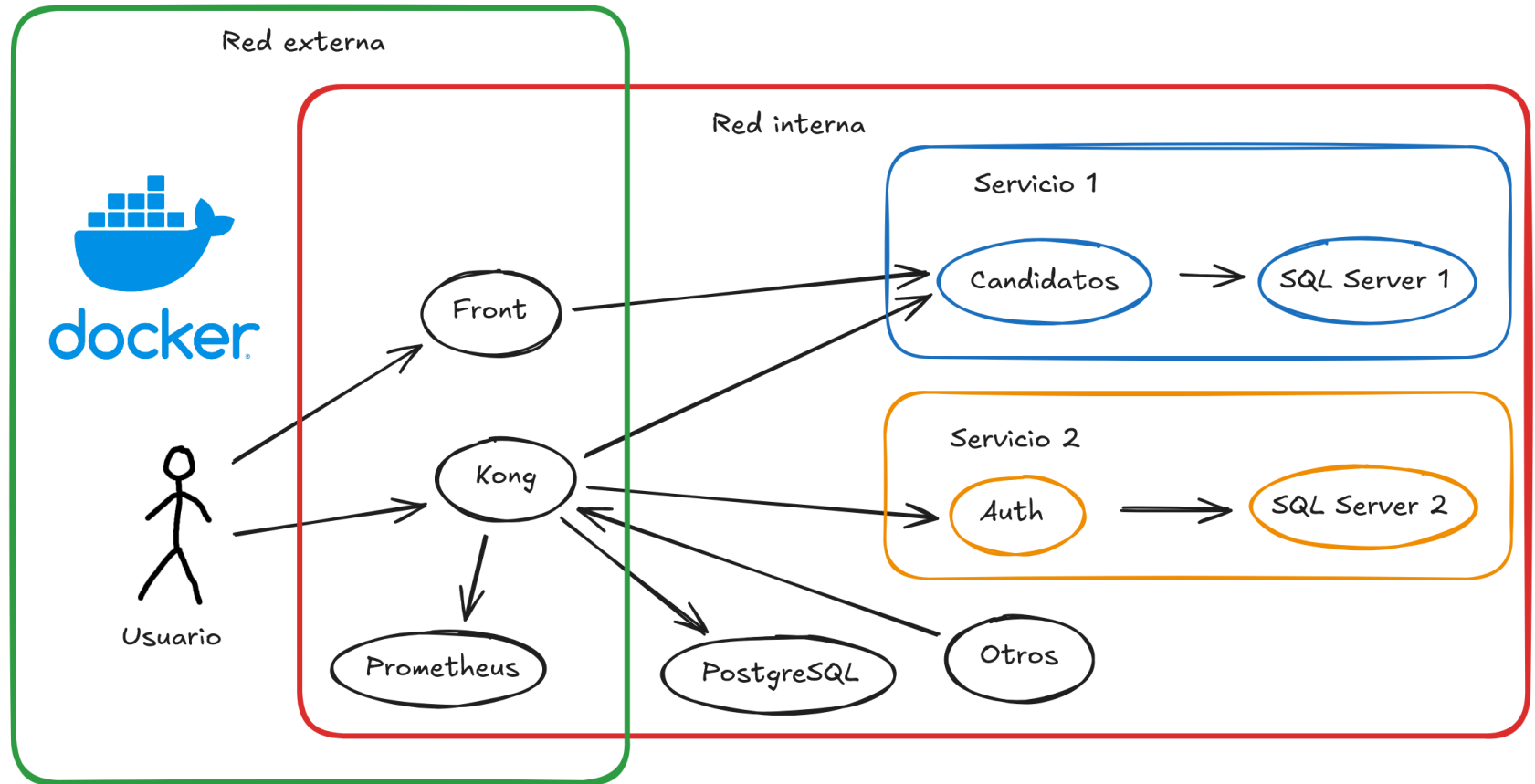
☒ Usuarios

☒ Comentarios

ARQUITECTURA DE LA SOLUCIÓN



COMUNICACIÓN ENTRE MICROSERVICIOS



DESPLIEGUE EN ENTORNO LOCAL

CONTINUACIÓN DEL RFI I



Una vez dockerizado el proyecto, el despliegue se se lleva a cabo haciendo uso de docker-compose.

Se despliegan de este modo todos los contenedores a la vez.

PASOS PARA DESPLIEGUE EN LOCAL USANDO DOCKER-COMPOSE

1. Situar .env en raíz del proyecto
2. `docker-compose up -d kong-migration`
3. `docker-compose up -d --build`
4. `docker-compose up -d kong-config`



Ejecutar una única vez

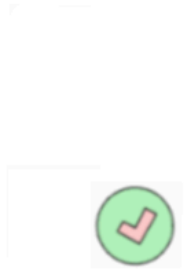
Para detener los contenedores: `docker-compose down`

Para ejecutar de nuevo los contenedores: `docker-compose up -d`

PROBAR FUNCIONALIDADES

BACKEND

Test Unitarios Backend



POSTMAN

- **Documentación interactiva**
- **Pruebas Rápidas y Eficientes**
- **Simulación de Respuestas**
- **Facilidad de Integración**
- **Interfaz de usuario simple y sencilla**

Candidates ^		
GET	/api/Candidates/Test	▼
GET	/api/Candidates/ObtenerTodosCandidatos	▼
GET	/api/Candidates/ObtenerCandidatoPorId/{id}	▼
POST	/api/Candidates/InsertarNuevoCandidato	▼
PUT	/api/Candidates/ActualizarVotosCandidato	▼
DELETE	/api/Candidates/EliminarCandidato/{id}	▼
GET	/api/Candidates/ObtenerVotosCandidato/{id}	▼
Authorization ^		
POST	/api/Auth/Login	▼
POST	/api/Auth/Register	▼

PROBAR FUNCIONALIDADES

FRONTEND

Test Unitarios Frontend



- **Voting Functionality**
- **Update Charts**
- **Render Charts**
- **Menu Section Switching**
- **Icon Circle Click**
- **Get Bar Chart Data**
- **Get Pie Chart Data**
- **Get General Chart Data**
- **Fetch Call**

AUTORIZACIÓN, AUTENTICACIÓN Y AUD.

AUTENTICACIÓN

- Empleo de servicio de Autenticación para obtener token
- Verificación de usuario y contraseña
- Token validez 1 hora
- JWT: Estándar para autenticación
- Rol: Usuario normal



AUTORIZACIÓN

- Kong (Api Gateway) valida token
- Rol Admin: Permite todas las llamadas
- Rol Normal: Permite solo consultas



AUDITORIA

- Plugin "file-log" para guardar logs de consultas
- Se crea uno por cada servicio
- Ubicación /tmp
- Kong tiene su propia BD



SOLUCIÓN ESCALABLE Y ESLÁSTICA

✓ Arquitectura escalable y elástica

Enfoque por capas:

Frontend, APIs y base de datos escalan de forma independiente

Servicios desacoplados:

Escalado horizontal o vertical según la demanda

Uso de contenedores:

Docker facilita despliegues portables y gestionados con Docker Compose

-> Aporta escalabilidad

-> Aporta elasticidad

CI/CD con GitHub Actions:

Integración y despliegue continuo para adaptabilidad

-> Se ajusta a nuevas demandas

◆ Resultado: Solución flexible, eficiente y preparada para crecimiento

Demo