### ****1. Arquitectura general en AWS (Pendiente)****

* **Servicios Auth y Chat**: Definidos en los microservicios **Auth Service** y **Chat Service**.
* **Decisión entre RDS o DynamoDB (para datos)**: Pendiente.
* **Decisión entorno de despliegue (ECS, Elastic Beanstalk, etc.)**: Pendiente.

### ****2. Microservicio de Autenticación (Auth Service)**** ✅

* **Proyecto Spring Boot con Spring Security, Spring Data JPA**:
  + Creado el proyecto con los paquetes: **model, controller, service, security, config**.
  + Endpoints básicos implementados:
    - **POST /auth/register**: Registro de usuario (recibe email y contraseña, almacena con contraseña hasheada).
    - **POST /auth/login**: Validación de credenciales (autentica usuario y devuelve token JWT).
  + **DTOs**: Implementados **UserDTO**, **LoginRequest**, **LoginResponse**, **RegisterRequest**.
  + **Validación JWT**:
    - **JwtUtils** para generar y validar el token.
    - **SecurityConfig** configurado para la seguridad.
    - **JwtAuthenticationFilter** para validar el token en los endpoints protegidos.

### ****3. Microservicio de Chat (Chat Service)**** ✅

* **Configuración de un microservicio con WebSockets**:
  + Implementado **WebSocket** en **ChatService**.
  + **Conexión WebSocket**: Endpoint para establecer la conexión y validar el token JWT en la conexión.
  + **Gestión de mensajes**: Implementada la lógica para enviar y recibir mensajes a través del WebSocket.

### ****4. Comunicación entre microservicios (Auth y Chat Services)**** ✅

* **Clave secreta compartida**: Configurada la validación del token JWT con la misma clave entre **Auth Service** y **Chat Service**.
* **Pruebas de comunicación**: Se realizaron pruebas con **Postman** y los dos microservicios funcionando en Spring Boot, y la comunicación fue exitosa (los servicios se autentican mutuamente).

### ****5. Despliegue en AWS (Pendiente)****

* **Contenedores Docker para Auth y Chat**: Pendiente.
* **Imágenes en AWS ECR**: Pendiente.
* **Cluster ECS o Elastic Beanstalk**: Pendiente.
* **Balanceador de carga (ALB)**: Pendiente.

### ****6. Frontend (Pendiente)****

* **Cliente en React/Angular/Vue**: Pendiente.
  + Consumir endpoints de **Auth Service** para login y registro.
  + Establecer conexión WebSocket para el chat.

### ****7. Seguridad y buenas prácticas (Parcialmente realizado)**** ✅

* **Almacenamiento de contraseñas hasheadas con BCrypt**: Implementado.
* **Validación JWT a nivel de código**: Implementado.
* **Pendientes**:
  + Configurar tráfico seguro (HTTPS).
  + Configurar AWS Secrets Manager o Parameter Store.
  + Implementar monitoreo con AWS CloudWatch/X-Ray.

### ****8. Testing (Implementado y Funcional)**** ✅

* **Pruebas unitarias**:
  + Se realizaron pruebas unitarias con **JUnit** para los endpoints en **Auth Service** y **Chat Service**.
  + Todos los tests fueron ejecutados con **mvn clean test**, y no se encontraron fallos en las pruebas, confirmando que la funcionalidad está operativa.
* **Pruebas de integración**:
  + Se probó la comunicación entre **Auth Service** y **Chat Service**, verificando que ambas aplicaciones se comunican correctamente.

### ****Resumen del progreso actual**** ✅

* **Auth Service**: Funcional con los endpoints de **registro**, **login**, y **validación de JWT**.
* **Chat Service**: Funcional con WebSockets y comunicación de mensajes.
* **Seguridad básica**: Implementada (contraseñas hasheadas, validación JWT).
* **Pruebas**: Unitarias y de integración realizadas con éxito.
* **Pendientes**: Despliegue en AWS, frontend, pruebas finales, y configuración adicional de seguridad.