Diagrama de Solución

Apolinar Solis Ordoñez

Correo: apolinar22@gmail.com

Celular: 3154018073

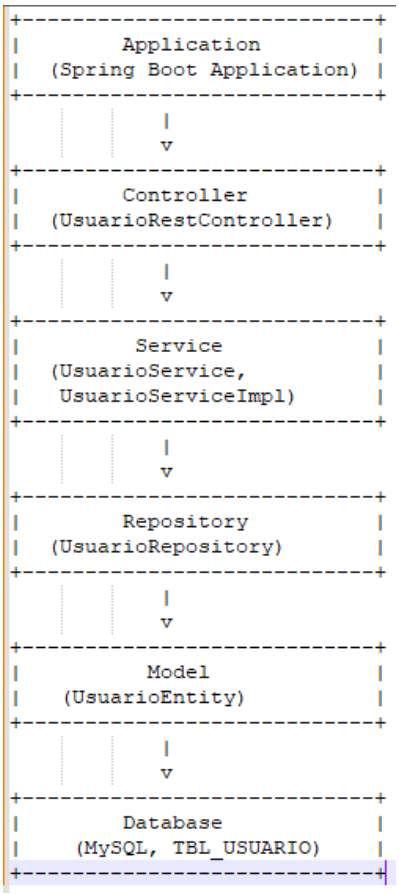
Cali - Colombia

El proyecto que describes está estructurado en varias capas siguiendo la arquitectura de software clásica de una aplicación web basada en Spring Boot. A continuación, se presenta un diagrama de alto nivel y la descripción de cada componente de la solución:

1. **Base de Datos (H2)**:
    - Tabla **TBL_USUARIO** que almacena los datos de los usuarios.
2. **Modelo (Entity)**:
    - **UsuarioEntity**: Clase que mapea la tabla **TBL_USUARIO** a una entidad Java con anotaciones de JPA.
3. **Repositorio (Repository)**:
    - **UsuarioRepository**: Interfaz que extiende **JpaRepository** y proporciona métodos para acceder y manipular los datos de **UsuarioEntity**.
4. **Servicio (Service)**:
    - **UsuarioService**: Interfaz que define los métodos del servicio.
    - **UsuarioServiceImpl**: Implementación de la interfaz **UsuarioService** que contiene la lógica de negocio.
    - **GenericService** y **GenericFechasService**: Interfaces genéricas para servicios con métodos comunes.
    - **ServiceException**: Clase que maneja las excepciones del servicio.
    - **UserDetailsServiceImpl**: Implementación de **UserDetailsService** para la autenticación.
5. **Controlador (Controller)**:
    - **UsuarioRestController**: Controlador REST que maneja las solicitudes HTTP para la entidad **UsuarioEntity**.
6. **Seguridad (Security)**:
    - Configuración de seguridad con JWT para la autenticación.
    - **JwtService**, **JwtServiceImpl**, **JwtAuthenticationFilter**, **SecurityConfiguration**: Clases que manejan la generación, validación y configuración de JWT.
7. **Excepciones (Exception)**:
    - **CorreoYaRegistradoException**: Excepción personalizada para manejar errores de correo ya registrado.
    - **GlobalExceptionHandler**: Manejador global de excepciones.

8. **Configuración de Aplicación**:
    - **Application.java**: Clase principal que inicializa la aplicación Spring Boot.

Diagrama:



Descripción de los Componentes:

1. **Application**:
    - **Application.java**: Punto de entrada de la aplicación Spring Boot.
2. **Controller**:
    - **UsuarioRestController**: Controlador que maneja las solicitudes HTTP (GET, POST, PUT, DELETE) para la entidad **UsuarioEntity**.
3. **Service**:
    - **UsuarioService**: Interfaz que define los métodos de negocio.
    - **UsuarioServiceImpl**: Implementación de **UsuarioService** que contiene la lógica de negocio para **UsuarioEntity**.
    - **GenericService**, **GenericFechasService**: Interfaces genéricas con métodos comunes para servicios.

- **ServiceException**: Clase de excepción para manejar errores del servicio.

4. **Repository**:
    - **UsuarioRepository**: Interfaz que extiende **JpaRepository** proporcionando métodos para operaciones CRUD sobre **UsuarioEntity**.
5. **Model**:
    - **UsuarioEntity**: Clase que mapea la tabla **TBL_USUARIO** a una entidad Java usando anotaciones JPA.
6. **Database**:
    - **TBL_USUARIO**: Tabla en la base de datos H2 que almacena los datos de los usuarios.
7. **Security**:
    - **JwtService**, **JwtServiceImpl**: Servicios para manejar la generación y validación de tokens JWT.
    - **JwtAuthenticationFilter**: Filtro para interceptar solicitudes HTTP y validar tokens JWT.
    - **SecurityConfiguration**: Configuración de seguridad de Spring Security.
    - **UserDetailsServiceImpl**: Implementación de **UserDetailsService** que carga los detalles del usuario para la autenticación.
8. **Exceptions**:
    - **CorreoYaRegistradoException**: Excepción personalizada para manejar el error de correo ya registrado.
    - **GlobalExceptionHandler**: Manejador global de excepciones que captura y responde a errores en las solicitudes HTTP.

Flujo de Trabajo:

1. **Solicitud HTTP**:
    - El cliente envía una solicitud HTTP al controlador **UsuarioRestController**.
2. **Controlador**:
    - **UsuarioRestController** procesa la solicitud y llama a los métodos del servicio **UsuarioService**.
3. **Servicio**:
    - **UsuarioServiceImpl** implementa la lógica de negocio y maneja las operaciones solicitadas.

- Interactúa con **UsuarioRepository** para realizar operaciones en la base de datos.

4. **Repositorio**:
    - **UsuarioRepository** interactúa con la base de datos para realizar operaciones CRUD sobre **UsuarioEntity**.
5. **Modelo**:
    - **UsuarioEntity** representa la estructura de la tabla **TBL_USUARIO** en la base de datos.
6. **Seguridad**:
    - **JwtAuthenticationFilter** valida las solicitudes HTTP utilizando JWT.
    - **JwtServiceImpl** genera y valida los tokens JWT.
    - **SecurityConfiguration** configura las reglas de seguridad.
7. **Excepciones**:
    - **CorreoYaRegistradoException** se lanza si se intenta registrar un correo ya existente.
    - **GlobalExceptionHandler** captura y maneja las excepciones, proporcionando respuestas HTTP adecuadas.

8. Este diagrama UML muestra cómo las distintas partes de la aplicación están conectadas entre sí, ayudando a visualizar la arquitectura general y las relaciones entre los componentes.

```
classDiagram
  class application.properties {
    +spring.application.name
    +server.port
    +spring.h2.console.enabled
    +spring.datasource.url
    +spring.datasource.driver-class-name
    +spring.datasource.username
    +spring.datasource.password
    +spring.jpa.database-platform
    +spring.jpa.defer-datasource-initialization
    +custom.igv
    +token.signing.key
  }

  class TBL_USUARIO {
```

```
    +ID
    +NOMBRE
    +USUARIO
    +CORREO
    +CLAVE
    +TELEFONO
    +ESTADO
    +CODIGO_CIUDAD
    +CODIGO_PAIS
    +FECHA_CREACION
    +FECHA_ULTIMA_ACTUALIZACION
    +FECHA_ULTIMO_INGRESO
}

class UsuarioEntity {
    -Long id
    -String nombre
    -String usuario
    -String correo
    -String clave
    -String telefono
    -String estado
    -String codigoCiudad
    -String codigoPais
    -Date fechaCreacion
    -LocalDateTime fechaUltimaActualizacion
    -LocalDateTime fechaUltimoIngreso
}

class UsuarioRepository {
    <<interface>>
    +Optional<UsuarioEntity> findByCorreo(String correo)
    +Optional<UsuarioEntity> findByUsuario(@Param("usuario") String usuario)
}

class GenericService {
    <<interface>>
    +Optional<T> findById(T t)
    +List<T> findLikeObject(T t)
    +T save(T t)
    +T update(T t)
    +Boolean delete(T t)
}

class UsuarioService {
    <<interface>>
}
```

```
  class UsuarioServiceImpl {
    -UsuarioRepository usuarioRepository
    +Optional<UsuarioEntity> findById(UsuarioEntity usuarioEntity)
    +UsuarioEntity save(UsuarioEntity usuarioEntity)
    +UsuarioEntity update(UsuarioEntity usuarioEntity)
    +Boolean delete(UsuarioEntity t)
    +List<UsuarioEntity> findLikeObject(UsuarioEntity t)
    +UsuarioEntity saveIfNotExists(UsuarioEntity entity)
  }

  class UsuarioRestController {
    -UsuarioService usuarioService
    +ResponseEntity<List<UsuarioEntity>> findAll()
    +ResponseEntity<UsuarioEntity> findById(@PathVariable("id") Long id)
    +ResponseEntity<List<UsuarioEntity>> findByLikeRazonSocial(@RequestParam(value =
"nombre") String nombre)
    +ResponseEntity<?> save(@RequestBody UsuarioEntity usuarioEntity)
    +ResponseEntity<UsuarioEntity> update(@PathVariable("id") Long id, @RequestBody
UsuarioEntity usuarioEntity)
    +ResponseEntity<Void> delete(@PathVariable("id") Long id)
  }

  class CorreoYaRegistradoException {
    +CorreoYaRegistradoException(String message)
  }

  class GlobalExceptionHandler {
    +ResponseEntity<String> handleCorreoYaRegistradoException(CorreoYaRegistradoException
ex)
  }

  class AppEncoder {
    +void main(String[] args)
  }

  class AuthenticationController {
    -AuthenticationService authenticationService
    +ResponseEntity<JwtAuthenticationDTOResponse> signin(@RequestBody LoginDTORequest
request)
  }

  class AuthenticationService {
    <<interface>>
    +JwtAuthenticationDTOResponse signin(LoginDTORequest request)
  }

  class AuthenticationServiceImpl {
    -UserDetailsService userDetailsService
```

```
    -JwtService jwtService
    -AuthenticationManager authenticationManager
    +JwtAuthenticationDTOResponse signin(LoginDTORequest request)
  }

  class JwtService {
    <<interface>>
    +String extractUser(String token)
    +String generateToken(UserDetails userDetails)
    +boolean isTokenValid(String token, UserDetails userDetails)
  }

  class JwtServiceImpl {
    -String jwtSigningKey
    +String extractUser(String token)
    +String generateToken(UserDetails userDetails)
    +boolean isTokenValid(String token, UserDetails userDetails)
  }

  class JwtAuthenticationFilter {
    -JwtService jwtService
    -UserDetailsService userDetailsService
    +void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain
filterChain)
  }

  class SecurityConfiguration {
    -JwtAuthenticationFilter jwtAuthenticationFilter
    -UserDetailsService userDetailsService
    +SecurityFilterChain securityFilterChain(HttpSecurity http)
    +AuthenticationProvider authenticationProvider()
    +AuthenticationManager authenticationManager(AuthenticationConfiguration config)
    +PasswordEncoder passwordEncoder()
  }

  class SecurityConstant {
    +Long EXPIRE
  }

  application.properties --> SecurityConfiguration
  application.properties --> JwtServiceImpl
  application.properties --> UsuarioRepository
  TBL_USUARIO --> UsuarioEntity
  UsuarioEntity --> UsuarioRepository
  UsuarioRepository --> UsuarioServiceImpl
  UsuarioServiceImpl --> UsuarioRestController
  UsuarioServiceImpl --> AuthenticationServiceImpl
  UsuarioRestController --> GlobalExceptionHandler
```

AuthenticationServiceImpl --> JwtServiceImpl
JwtAuthenticationFilter --> SecurityConfiguration
AuthenticationController --> AuthenticationServiceImpl
AppEncoder --> AuthenticationController