Buenas prácticas de desarrollo de software - Organización del Proyecto

Esta sección expone recomendaciones para organizar un proyecto de software. Como referencia se utilizará la tecnología java con el framework springboot.

Prácticas Recomendadas

Como práctica recomendada en la industria del desarrollo de software, se debe utilizar una estructura de carpetas bien organizada para mejorar la capacidad de mantenimiento del código, la colaboración entre los miembros del equipo y el proceso de desarrollo general.

A continuación se detallan algunas razones clave para organizar la estructura de carpetas de un protecto Spring Boot:

Claridad y organización

Una estructura de carpetas bien definida proporciona un diseño claro de dónde encontrar los diferentes componentes de su aplicación. Facilita a los desarrolladores la localización de archivos, paquetes y recursos específicos, lo que reduce el tiempo dedicado a buscar código.

Modularidad

Spring Boot sigue un enfoque modular en el que la aplicación se divide en diferentes componentes lógicos como controladores, servicios, repositorios, configuraciones, etc. Tener una estructura de carpetas adecuada le permite organizar estos componentes de manera efectiva, lo que hace que la aplicación sea más fácil de entender y mantener.

Separación de responsabilidades

La estructura de carpetas impone una separación de responsabilidades, lo que permite a los desarrolladores centrarse en partes específicas de la aplicación sin preocuparse por código no relacionado. Por ejemplo, la lógica empresarial reside en la capa de servicio, el código de acceso a datos en la capa de repositorio y el manejo de solicitudes web en la capa de controlador.

Escalabilidad

A medida que la aplicación crece, una estructura de carpetas bien organizada facilita la incorporación de nuevas características y funcionalidades sin convertir el proyecto en un desastre inmanejable. Facilita la adición de nuevos módulos o componentes manteniendo una arquitectura general coherente.

Reutilización del código

Una organización adecuada fomenta la creación de componentes reutilizables. Cuando los desarrolladores pueden encontrar y comprender fácilmente el código existente, es más probable que lo reutilicen en lugar de duplicar esfuerzos.

Facilidad de colaboración

Cuando varios desarrolladores trabajan en un proyecto, seguir una estructura de carpetas estandarizada garantiza la coherencia y facilita la colaboración. Todos los miembros del equipo saben dónde colocar el código nuevo y pueden localizar rápidamente el código agregado por otros. Compilación e implementación: herramientas como sistemas de compilación, integración continua y scripts de implementación pueden aprovechar una estructura de carpetas predecible para automatizar los procesos de compilación e implementación de manera eficiente.

Pruebas y depuración

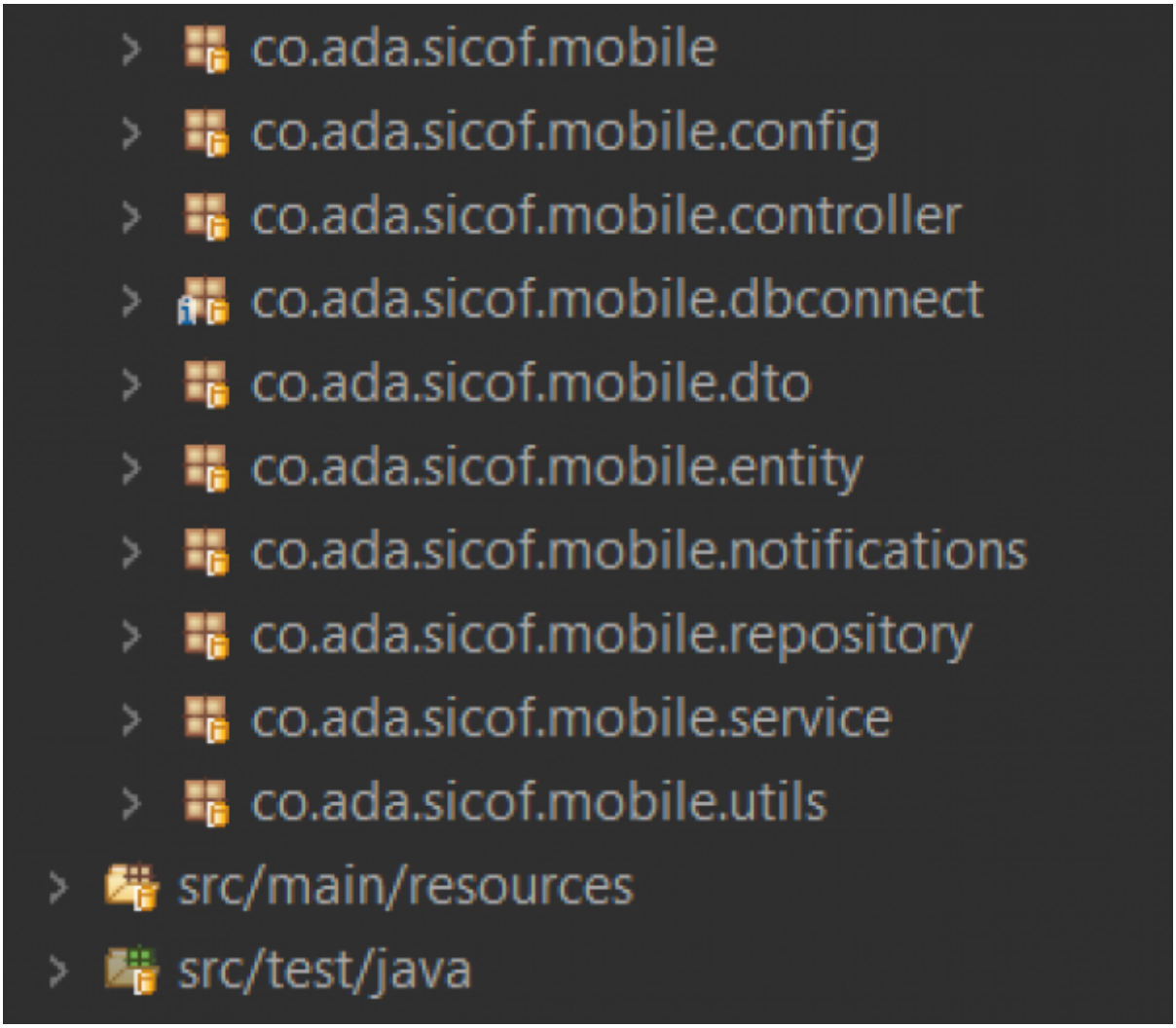
Una estructura de carpetas organizada simplifica la escritura de pruebas y la depuración de la aplicación. Las clases y recursos de prueba se pueden colocar junto a las clases correspondientes, lo que facilita la comprensión de la cobertura del conjunto de pruebas.

Incorporación de nuevos miembros del equipo

Cuando nuevos desarrolladores se unen al equipo, una estructura de carpetas bien definida les ayuda a ponerse al día rápidamente. Pueden comprender rápidamente la arquitectura y localizar secciones de código relevantes. Base de código mantenible: una base de código limpia y bien estructurada es más fácil de mantener a largo plazo. Reduce la deuda técnica y hace que sea menos probable que se introduzcan errores o regresiones al realizar cambios.

Ejemplo práctico

A continuación se comparte el siguiente gráfico de referencia que proponen una estructura de organización de un proyecto springboot.



Directorio: Config

Contiene clases de configuración, donde configura los ajustes de la aplicación o otras configuraciones a nivel de aplicación.

```
/**
 * Copyright © 2023 ADA Corporation. All rights reserved.
 *
 * This component is protected by copyright.
 *
 * Use of this component is subject to the terms of the license agreement.
 */
package com.ada.viatico.proceso.config;

import java.util.List;
import java.util.TimeZone;

import javax.annotation.PostConstruct;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.Configuration;
import
org.springframework.context.support.PropertySourcesPlaceholderConfigurer;

import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.info.Contact;
import io.swagger.v3.oas.models.info.Info;
import io.swagger.v3.oas.models.info.License;
import io.swagger.v3.oas.models.servers.Server;

@Configuration
public class ViaticosConfig {

    /** The log. */
    private static Logger log =
LoggerFactory.getLogger(ViaticosConfig.class);

    /** The url. */
    @Value("${openapi.url}")
    private String url;

    /** The application title. */
    @Value("${application.title}")
    private String applicationTitle;

    /** The application summary. */
    @Value("${application.version}")
    private String applicationVersion;

    @PostConstruct
    public void init() {
        log.info("Establicendo Zona Horaria...");
        TimeZone.setDefault(TimeZone.getTimeZone("America/Bogota"));
        log.info("Zona Horaria establecida a {}",
TimeZone.getDefault().getID());
    }

    /**
     * Property sources placeholder configurer.
     *
     * @return the property sources placeholder configurer
     */
    @Bean
    public static PropertySourcesPlaceholderConfigurer
propertySourcesPlaceholderConfigurer() {
        return new PropertySourcesPlaceholderConfigurer();
    }

    /**
     * My open API.
```

```
     *
     * @return the open API
     */
    @Bean
    public OpenAPI myOpenAPI() {
        Server server = new Server();
        server.setUrl(url);
        server.setDescription("Server URL environment");

        Contact contact = new Contact();
        contact.setEmail("carlos.torres@ada.co; daniel.lopez@ada.co");
        contact.setName("Carlos Torres | Daniel López");
        contact.setUrl("www.ada.co");

        License mitLicense = new License().name("ADA
License").url("www.ada.co/licenses/");

        Info info = new Info()
                .title(applicationTitle)
                .version("v" + applicationVersion)
                .summary("Microservicio para gestionar Viáticos.")
                .description("Servicio que realiza las operaciones de
creación/aprobación de Solicitudes y Legalización de Viáticos.")
                .contact(contact)
                .termsOfService("https://www.ada.co/terms")
                .license(mitLicense);

        return new OpenAPI().info(info).servers(List.of(server));
    }

}
```

Directorio: Controller

Contiene sus clases de controlador RESTful. Estas clases manejan solicitudes HTTP entrantes y definen los puntos finales de la API.

```
/**
 * Copyright © 2023 ADA Corporation. All rights reserved.
 *
 * This component is protected by copyright.
 *
 * Use of this component is subject to the terms of the license agreement.
 */
package com.ada.viatico.proceso.controller.v2;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
```

```
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.ada.viatico.proceso.dto.DatosGenerarAprobacion;
import com.ada.viatico.proceso.dto.GenericResponseDto;
import com.ada.viatico.proceso.repository.IMestroCompromisoRepo;
import com.ada.viatico.proceso.service.Impl.TicketsImp;
import com.ada.viatico.proceso.service.Impl.ViaticoSolicitudImpl;
import com.ada.viatico.proceso.service.v2.ViaticosServiceV2;
import com.ada.viatico.proceso.sfcomercial.dto.AprobacionLegalizacionDto;
import com.ada.viatico.proceso.utiliti.AdelantoSolicitud;
import com.ada.viatico.proceso.utiliti.GuardarLegalizacion;
import com.ada.viatico.proceso.utiliti.ProcesoGuardar;
import com.ada.viatico.proceso.utiliti.ProcesoViaticoAprobacion;
import com.ada.viatico.proceso.utiliti.ViaticosException;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.enums.ParameterIn;
import io.swagger.v3.oas.annotations.media.Content;
import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.responses.ApiResponse;

/**
 * The Class ViaticosController.
 * @version 2.0
 * @author carlos.torres@ada.co
 */
@CrossOrigin(origins = "*", methods = {RequestMethod.GET,
RequestMethod.POST, RequestMethod.DELETE})
@RestController
@RequestMapping("v2")
public class ViaticosControllerV2 {

    /** The log. */
    private Logger log =
LoggerFactory.getLogger(ViaticosControllerV2.class);

    /** The ticketsimpl. */
    @Autowired
    TicketsImp ticketsimpl;

    /** The guardar. */
    @Autowired
    ProcesoGuardar guardar;
```

```
    /** The aprobacion. */
    @Autowired
    ProcesoViaticoAprobacion aprobacion;

    /** The solicitud repo. */
    @Autowired
    ViaticoSolicitudImpl solicitudRepo;

    /** The savelega. */
    @Autowired
    GuardarLegalizacion savelega;

    /** The repo compromiso. */
    @Autowired
    IMestroCompromisoRepo repoCompromiso;

    /** The solicitud adelanto. */
    @Autowired
    AdelantoSolicitud solicitudAdelanto;

    /** The viaticos service. */
    @Autowired
    ViaticosServiceV2 viaticosServiceV2;

    /**
     * Aprobar legalizacion.
     *
     * @param aprobacionLegalizacionDto the aprobacion legalizacion dto
     * @return the response entity
     * @throws ViaticosException the viaticos exception
     */
    @Operation(summary = "Aprobar Legalización.", description = "Aprueba la
Legalización de un Viático.")
    @ApiResponse(responseCode = "201", content = { @Content(schema =
@Schema(implementation = GenericResponseDto.class)) })
    @ApiResponse(responseCode = "500", content = { @Content(schema =
@Schema(implementation = ViaticosException.class)) })
    @Parameter(name = "aprobacionLegalizacionDto", description = "Componente
que contiene los parametros necesarios para el proceso de legalización.", in
= ParameterIn.QUERY, required = true)
    @PostMapping("/aprobarlegalizacion")
    public ResponseEntity<GenericResponseDto>
aprobarLegalizacion(@RequestBody AprobacionLegalizacionDto
aprobacionLegalizacionDto) throws ViaticosException {
        var result =
viaticosServiceV2.aprobarLegalizacion(aprobacionLegalizacionDto);
        if(result.getCode() == -1L) {
            log.info("approve legalization executed with errors.");
            return new ResponseEntity<>(result,
HttpStatus.INTERNAL_SERVER_ERROR);
```

```
        }
        log.info("approve legalization executed correctly.");
        return new ResponseEntity<>(result, HttpStatus.CREATED);
    }

    /**
     * Guardar viatico.
     *
     * @param datos the datos
     * @return the response entity
     * @throws ViaticosException the viaticos exception
     */
    @Operation(summary = "Aprobar Solicitud.", description = "Aprueba la
solicitud de un Viático.")
    @ApiResponse(responseCode = "201", content = { @Content(schema =
@Schema(implementation = GenericResponseDto.class)) })
    @ApiResponse(responseCode = "500", content = { @Content(schema =
@Schema(implementation = ViaticosException.class)) })
    @Parameter(name = "datos", description = "Componente que contiene los
parametros necesarios para el proceso de solicitud.", in =
ParameterIn.QUERY, required = true)
    @PostMapping("/generaraprobacion")
    public ResponseEntity<GenericResponseDto> guardarViatico(@RequestBody
DatosGenerarAprobacion datos) throws ViaticosException {
        var result = viaticosServiceV2.aprobarSolicitud(datos);
        if(result.getCode() == -1L) return new ResponseEntity<>(result,
HttpStatus.SERVICE_UNAVAILABLE);
        return new ResponseEntity<>(result, HttpStatus.CREATED);
    }
}
```

Directorio: DBConnect

Contiene sus clases para permitir conexiones multicliente y modelo multiempresa.

Directorio: DTO(Objeto de transferencia de datos)

Un DTO es un patrón de diseño utilizado para transferir datos entre diferentes capas o componentes de una aplicación. El objetivo principal de un DTO es encapsular datos y proporcionar una estructura de datos simple que pueda transmitirse fácilmente por la aplicación. Los DTO se utilizan a menudo para transferir datos entre el front-end y el back-end de una aplicación web, entre microservicios o entre diferentes capas de una aplicación, como la capa de servicio y la capa de presentación.

Características de una DTO

- Por lo general, contiene solo campos privados con captadores y definidores para acceder a los

datos.

- Los DTO no contienen ninguna lógica empresarial y su objetivo principal es transportar datos.
- A menudo se utilizan para representar un subconjunto de datos de una entidad o una combinación de datos de varias entidades.
- Los DTO ayudan a reducir la cantidad de datos transferidos a través de la red, mejorando el rendimiento al evitar el intercambio excesivo de datos.

```java
/**
 * Copyright © 2023 ADA Corporation. All rights reserved.
 *
 * This component is protected by copyright.
 *
 * Use of this component is subject to the terms of the license agreement.
 */
package com.ada.viatico.proceso.sfcomercial.dto;

import java.io.Serializable;

import io.swagger.v3.oas.annotations.media.Schema;
import lombok.Data;

/**
 * Instantiates a new afectacion presupuestal dto.
 */
@Schema(description = "Dto que contiene la configuración de afectación
presupuestal de un rubro de una disponibilidad..")
@Data
public class AfectacionPresupuestalDto implements Serializable {

    /** The Constant serialVersionUID. */
    private static final long serialVersionUID = -7167570415608375973L;

    /** The cod disponibilidad. */
    @Schema(description = "Código de la disponibilidad.")
    Long codDisponibilidad;

    /** The codigo rubro. */
    @Schema(description = "Código del rubro.")
    Long codigoRubro;

    /** The valor. */
    @Schema(description = "Valor de afectación del rubro prespuestal.")
    Long valor;

    /** The cod C costos. */
    @Schema(description = "Código del centro de costos.")
    Long codCCostos;
}
```

Directorio: Entity / Model

La carpeta del modelo almacena modelos de datos o entidades que representan la estructura y el comportamiento del dominio de la aplicación. Estas clases se asignan a tablas, vistas o consultas de bases de datos y definen las propiedades y relaciones de los datos de la aplicación.

```
/**
 * Copyright © 2023 ADA Corporation. All rights reserved.
 *
 * This component is protected by copyright.
 *
 * Use of this component is subject to the terms of the license agreement.
 */
package com.ada.viatico.proceso.entity;

import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;

/**
 * Instantiates a new comprobante de egreso.
 */
@Data
@Entity
@Table(schema = "TESORE01",name = "COMPROBANTE_DE_EGRESO")
public class ComprobanteDeEgreso {

    /** The cod inter comprobante egreso. */
    @Id
    @Column(name="COD_INTER_COMPROBANTE_EGRESO")
    private Long codInterComprobanteEgreso;

    /** The consecutivo. */
    @Column(name="CONSECUTIVO")
    private Long consecutivo;

    /** The tipo. */
    @Column(name="TIPO")
    private String tipo;

    /** The estado. */
    @Column(name="ESTADO")
    private String estado;
```

```
    /** The fecha elaboracion. */
    @Column(name="FECHA_ELABORACION")
    private Date fechaElaboracion;

    /** The fecha impresion. */
    @Column(name="FECHA_IMPRESION")
    private Date fechaImpresion;
}
```

Directorio: Repository

Contiene clases de repositorio que se ocupan del acceso a datos. Estas clases suelen utilizar un marco ORM (Object-Relational Mapping) o JPA (Java Persistence API) para interactuar con la base de datos.

```
/**
 * Copyright © 2023 ADA Corporation. All rights reserved.
 *
 * This component is protected by copyright.
 *
 * Use of this component is subject to the terms of the license agreement.
 */
package co.ada.sicof.mobile.repository;

import co.ada.core.db.config.repository.JpaAdaRepository;
import co.ada.sicof.mobile.entity.Empresa;

/**
 * The Interface EmpresaRepository.
 */
public interface EmpresaRepository extends JpaAdaRepository<Empresa, Long> {

    /**
     * Find by codigo empresa.
     *
     * @param codigoEmpresa the codigo empresa
     * @return the empresa
     */
    public Empresa findByCodigoEmpresa(Long codigoEmpresa);

    /**
     * Find by codigo mempresa.
     *
     * @param codigoMempresa the codigo mempresa
     * @return the empresa
     */
    public Empresa findByCodigoMempresa(String codigoMempresa);
}
```

Directorio: Service

Contiene clases de servicio que implementan la lógica empresarial. Los controladores utilizan estos servicios para realizar operaciones con datos.

```
/**
 * Copyright © 2023 ADA Corporation. All rights reserved.
 *
 * This component is protected by copyright.
 *
 * Use of this component is subject to the terms of the license agreement.
 */
package com.ada.viatico.proceso.service.v2;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.ada.viatico.proceso.component.LegalizacionComponentV2;
import com.ada.viatico.proceso.component.SolicitudComponentV2;
import com.ada.viatico.proceso.dto.DatosGenerarAprobacion;
import com.ada.viatico.proceso.dto.GenericResponseDto;
import com.ada.viatico.proceso.repository.ITmpCompromisoRepository;
import com.ada.viatico.proceso.service.Impl.TicketsImp;
import com.ada.viatico.proceso.service.Impl.ViaticoSolicitudImpl;
import com.ada.viatico.proceso.sfcomercial.dto.AprobacionLegalizacionDto;
import com.ada.viatico.proceso.utiliti.ViaticosException;

/**
 * The Class ViaticosServiceV2.
 * @version 2.0
 */
@Service
public class ViaticosServiceV2 {

    /** The log. */
    Logger log = LoggerFactory.getLogger(ViaticosServiceV2.class);

    /** The Constant APROBRAR. */
    private static final String APROBRAR = "A";

    /** The ticketsimpl. */
    @Autowired
    private TicketsImp ticketsimpl;

    /** The solicitud repo. */
    @Autowired
```

```
    private ViaticoSolicitudImpl solicitudRepo;

    /** The compromiso tmp. */
    @Autowired
    public ITmpCompromisoRepository compromisoTmp;

    /** The legalizacion component V 2. */
    @Autowired
    private LegalizacionComponentV2 legalizacionComponentV2;

    /** The solicitud component. */
    @Autowired
    private SolicitudComponentV2 solicitudComponentV2;

    /**
     * Aprobar solicitud.
     *
     * @param datos the datos
     * @return the generic response dto
     * @throws ViaticosException the viaticos exception
     */
    @Transactional(rollbackFor = ViaticosException.class)
    public GenericResponseDto aprobarSolicitud(DatosGenerarAprobacion datos)
throws ViaticosException {
        if (datos.getEstadoSolicitudTipo().equals(APROBRAR)) {
            return solicitudComponentV2.realizarAprobacion(datos);
        } else {
            return solicitudComponentV2.cancelarAprobacion(datos);
        }
    }

    /**
     * Aprobar legalizacion.
     *
     * @param aprobacionLegalizacionDto the aprobacion legalizacion dto
     * @param viaticoDetalleSolicitud the viatico detalle solicitud
     * @param viaticoDetalleAprobacion the viatico detalle aprobacion
     * @return the generic response dto
     * @throws ViaticosException the viaticos exception
     */
    @Transactional(rollbackFor = ViaticosException.class)
    public GenericResponseDto aprobarLegalizacion(AprobacionLegalizacionDto
aprobacionLegalizacionDto) throws ViaticosException {
        if(aprobacionLegalizacionDto.getEstadoTipo().equals("A")) {
            GenericResponseDto genericResponseDto =
legalizacionComponentV2.generarProcesoInterfaz(aprobacionLegalizacionDto);
            if(genericResponseDto.getCode() == 1L) {
                solicitudRepo.generarUpdateEstadoTipo("A",
aprobacionLegalizacionDto.getIdTicket());
                ticketsimpl.generarUpdateEstadoTicket("A",
aprobacionLegalizacionDto.getIdTicketLegalizado(),
```

```
aprobacionLegalizacionDto.getDescripcion());
                compromisoTmp.actualizarOrdenador(0L,
aprobacionLegalizacionDto.getIdTicket());
            }
            return genericResponseDto;
        }else {
            solicitudRepo.generarUpdateEstadoTipo("V",
aprobacionLegalizacionDto.getIdTicket());
            ticketsimpl.generarUpdateEstadoTicket("V",
aprobacionLegalizacionDto.getIdTicketLegalizado(),
aprobacionLegalizacionDto.getDescripcion());
compromisoTmp.actualizarOrdenador(aprobacionLegalizacionDto.getKaNlTerceroFu
ncionario(),aprobacionLegalizacionDto.getIdTicket());
        }
        return new GenericResponseDto(1L, "El documento sigue pendiente por
proceso de aprobación.");
    }
}
```

Directorio: Utils(Utilidades)

Es un directorio general donde se registran clases de utilidad para mantener la base de código organizada y modular.

```
/**
 * Copyright © 2023 ADA Corporation. All rights reserved.
 *
 * This component is protected by copyright.
 *
 * Use of this component is subject to the terms of the license agreement.
 */
package com.ada.viatico.proceso.sfcomercial;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Date;

/**
 * The Class Utility.
 */
public class Utility {

    /** The instance. */
    private static Utility instance;

    /** The Constant colombia. */
```

```
    // Create a new Locale
    //public static final Locale colombia =new Locale("es", "CO");

    /** The Constant pesoFormat. */
    // Create a formatter given the Locale
    //public static final NumberFormat pesoFormat =
NumberFormat.getCurrencyInstance(colombia);

    public static final Long PRESUPUESTO=1L;
    public static final Long TESORERIA=2L;
    public static final Long CONTABILIDAD=2L;


    public static final Long COMPROMISO=2L;
    public static final Long COMPROBANTE_EGRESO=1L;
    public static final Long COMPROBANTE_INGRESO=2L;

    /**
     * Instantiates a new utility.
     */
    private Utility() {
    }

    /**
     * Gets the single instance of Utility.
     *
     * @return single instance of Utility
     */
    public static Utility getInstance() {
        if (instance == null)
            instance = new Utility();
        return instance;
    }

    /**
     * Convert from util date.
     *
     * @param utilDate the util date
     * @return the java.sql. date
     */
    public static java.sql.Date convertFromUtilDate(java.util.Date utilDate)
{
        if (utilDate == null)
            return null;
        return new java.sql.Date(utilDate.getTime());
    }

    /**
     * Convert from sql date.
     *
     * @param sqlDate the sql date
```

```
     * @return the java.util. date
     */
    public static java.util.Date convertFromSqlDate(java.sql.Date sqlDate) {
        if (sqlDate == null)
            return null;
        return new java.util.Date(sqlDate.getTime());
    }

    /**
     * Gets the date without time using format.
     *
     * @return the date without time using format
     * @throws ParseException the parse exception
     */
    public static Date getDateWithoutTimeUsingFormat() throws ParseException
{
        SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy");
        // Formatear la fecha actual sin hora y convertirla a Date
        return formatter.parse(formatter.format(new Date()));
    }

    /**
     * Gets the numero.
     *
     * @return the numero
     */
    public static String getNumero() {
        String formato = "yyyyMMddHHmmss";
        DateTimeFormatter formateador =
DateTimeFormatter.ofPattern(formato);
        LocalDateTime ahora = LocalDateTime.now();
        return formateador.format(ahora);
    }
}
```

Otros Directorios

Además de las carpetas src/main/java, hay otras carpetas que deben tenerse en una aplicación Springboot.

- **src/main/resources**: esta carpeta contiene recursos que no son Java, como archivos estáticos, plantillas y archivos de configuración.
- **src/test**: esta carpeta contiene todas sus clases de prueba. Dentro de esta carpeta hay otra carpeta que es igual a la estructura de carpetas *src/main/java*. Como ejemplo, la carpeta *src/test/java/service* contiene clases de prueba para probar las clases de servicio de las clases *src/main/java/service*.

Nota

La estructura de carpetas puede variar de una empresa a otra. Pero esta es la estructura de carpetas básica de una aplicación Spring Boot que se recomienda se implemente en la mayoria de proyectos.

←Volver atras

Contenido adaptado desde el sitio web:
https://malshani-wijekoon.medium.com/spring-boot-folder-structure-best-practices-18ef78a81819