Universidad Autónoma del Estado de México
Facultad de ingeniería

Documentación técnica
Sistema de cálculo de nómina

Saúl Tonatiúh
González Olivares

11 de diciembre de 2025

Tabla de contenido

1. Planificación

1.1. Diagnóstico

Necesidad de un sistema de login simple para un administrador el cual tendrá la posibilidad de registrar empleados y calcular la nómina de cada uno de ellos.

1.2. Requerimientos

1.2.1. Funcionales

- Crear una plataforma web.
- Crear un sistema de login.
- Un usuario puede hacer login con su usuario y contraseña.
- Un usuario puede registrar un empleado, ingresando su nombre, apellidos, RFC, y correo electrónico, en caso de ser un administrador o usuario, se debe especificar una contraseña.
- Calcular la nómina del empleado, para ello, se debe desplegar información esencial del empleado y se debe ingresar el monto de la nómina mensual.
- El sistema debe calcular la nómina correctamente basándose en las tablas del ISR oficiales del 2025 propias del SAT.
- El sistema mostrará el resultado de los cálculos: salario bruto, ISR correspondiente, deducciones y salario neto.
- El usuario puede cerrar su sesión

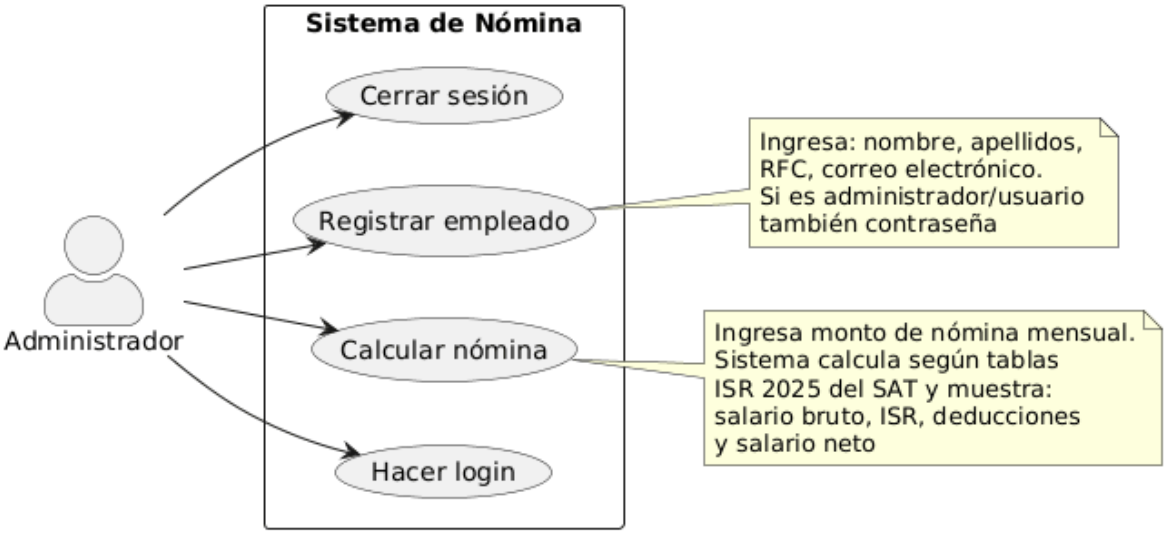
1.2.2. No funcionales

- Persistencia de los datos en una base de datos local.
- Seguridad en el inicio de sesión, no se debe permitir el acceso a las rutas que requieran con acceso restringido.
- Tiempo de ejecución y desempeño suficientes.

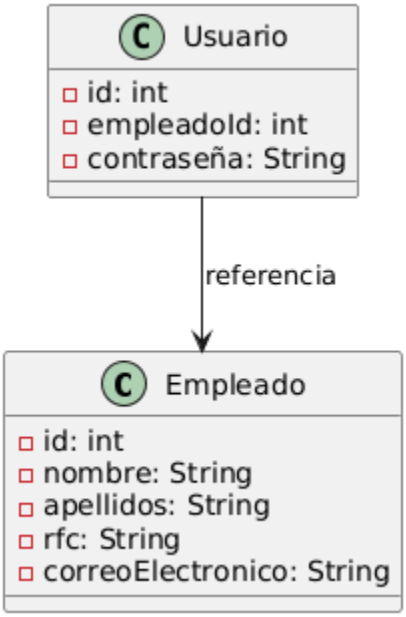
2. Diseño

En base a los requerimientos del proyecto, se diseñaron los siguientes diagramas, que ilustran el funcionamiento del sistema, las operaciones esenciales, y el flujo de la información y los datos.

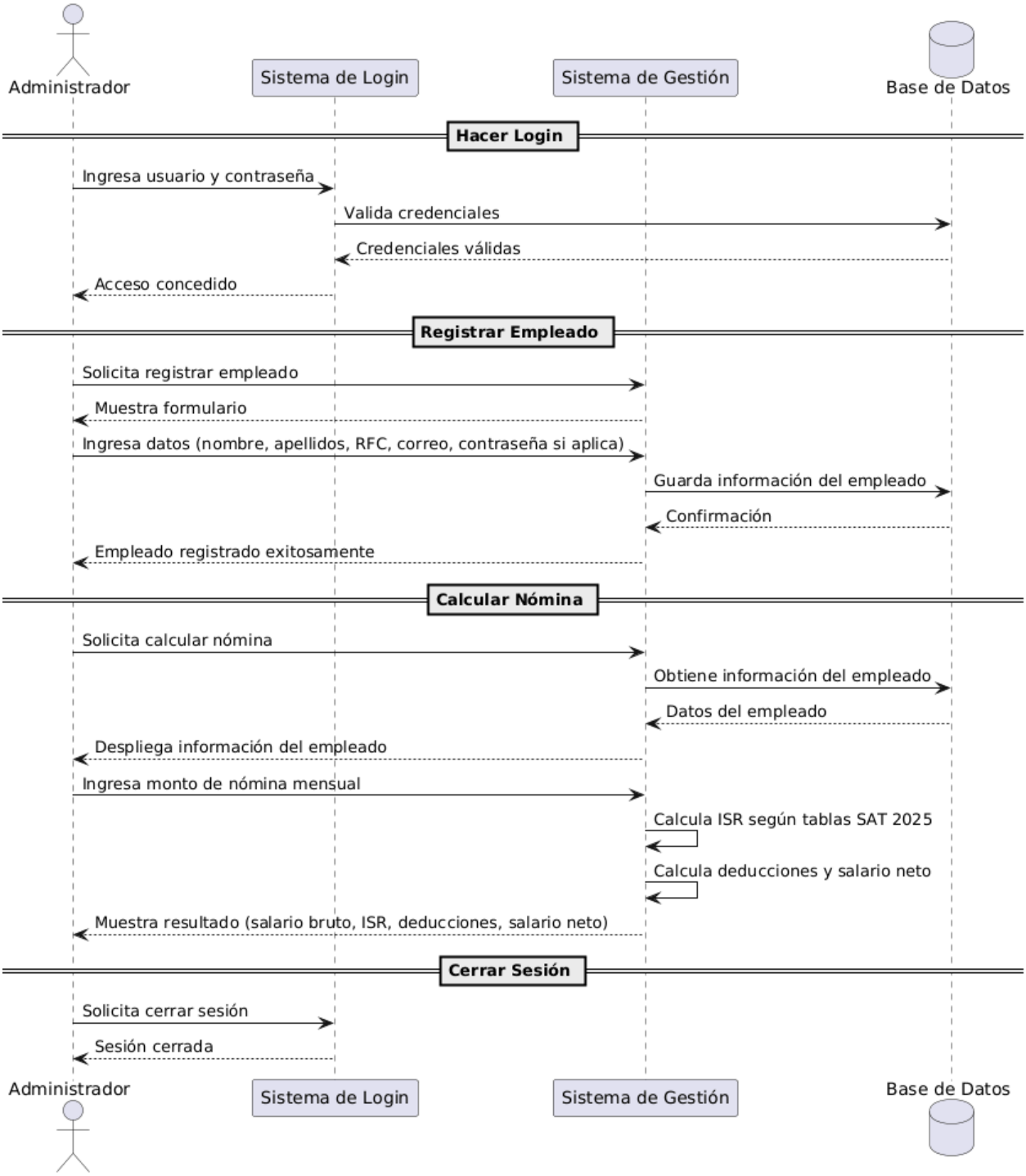
2.1. Diagrama de casos de uso



2.2. Diagrama de entidad-relación



2.3. Diagrama de secuencia



2.4. Tecnologías

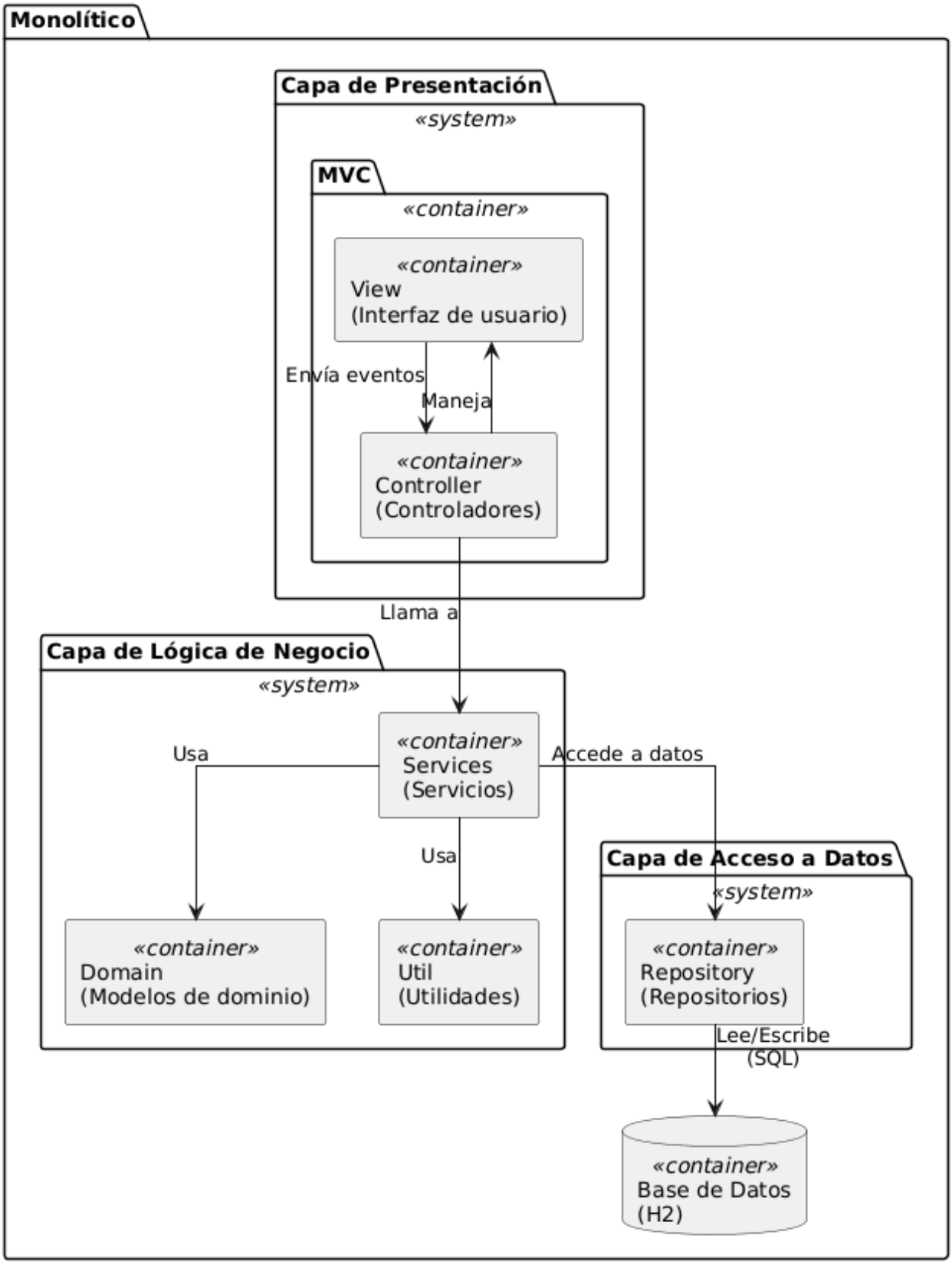
El sistema será construido con base en las siguientes tecnologías y herramientas de software.

- POO.
- Paradigma MVC.
- Java 17.0.12
- HTML, CSS y JS para el frontend.
- Framework de java **Spring Boot,** tanto para operaciones del backend como para despliegue y manejo de frontend.
- Apache Maven 3.9.11.
- Apache Tomcat 10.1.48.
- Hibernate.
- Thymeleaf.
- H2.

2.5. Arquitectura

El modelo de una página web se diseña y estructura particularmente con el **paradigma MVC** (modelo, vista, controlador) el cual nos permite separar las diversas capas del sistema para mejorar su seguridad y mantenimiento. A su vez, como se trata de un sistema sencillo, se trabajará sobre una **arquitectura monolítica de tipo cliente-servidor**, en la que se diferenciarán claramente las capas de presentación, de negocios, de acceso a datos y de persistencia de los datos.
La arquitectura general se basa en la de proyectos de java y particularmente proyectos en Spring boot, donde las capas y segmentación de paquetes está definida de mejor manera. Se presenta a continuación:

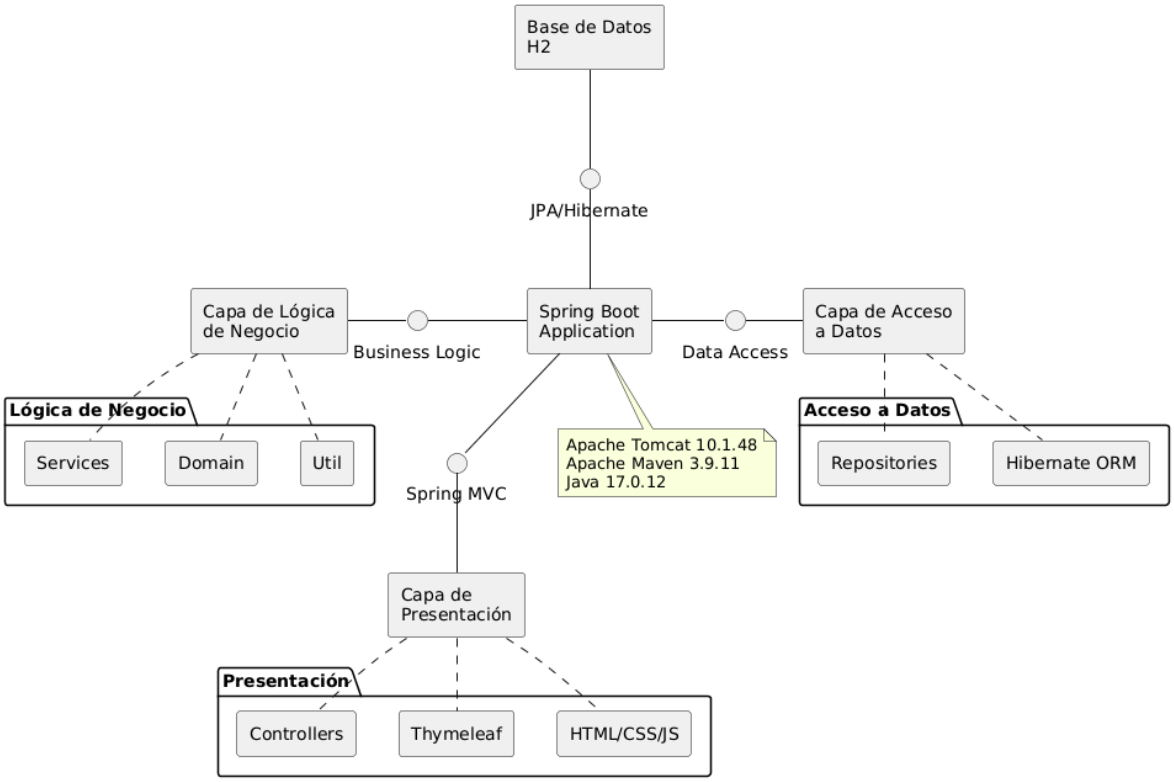
2.5.1. Diagrama del sistema



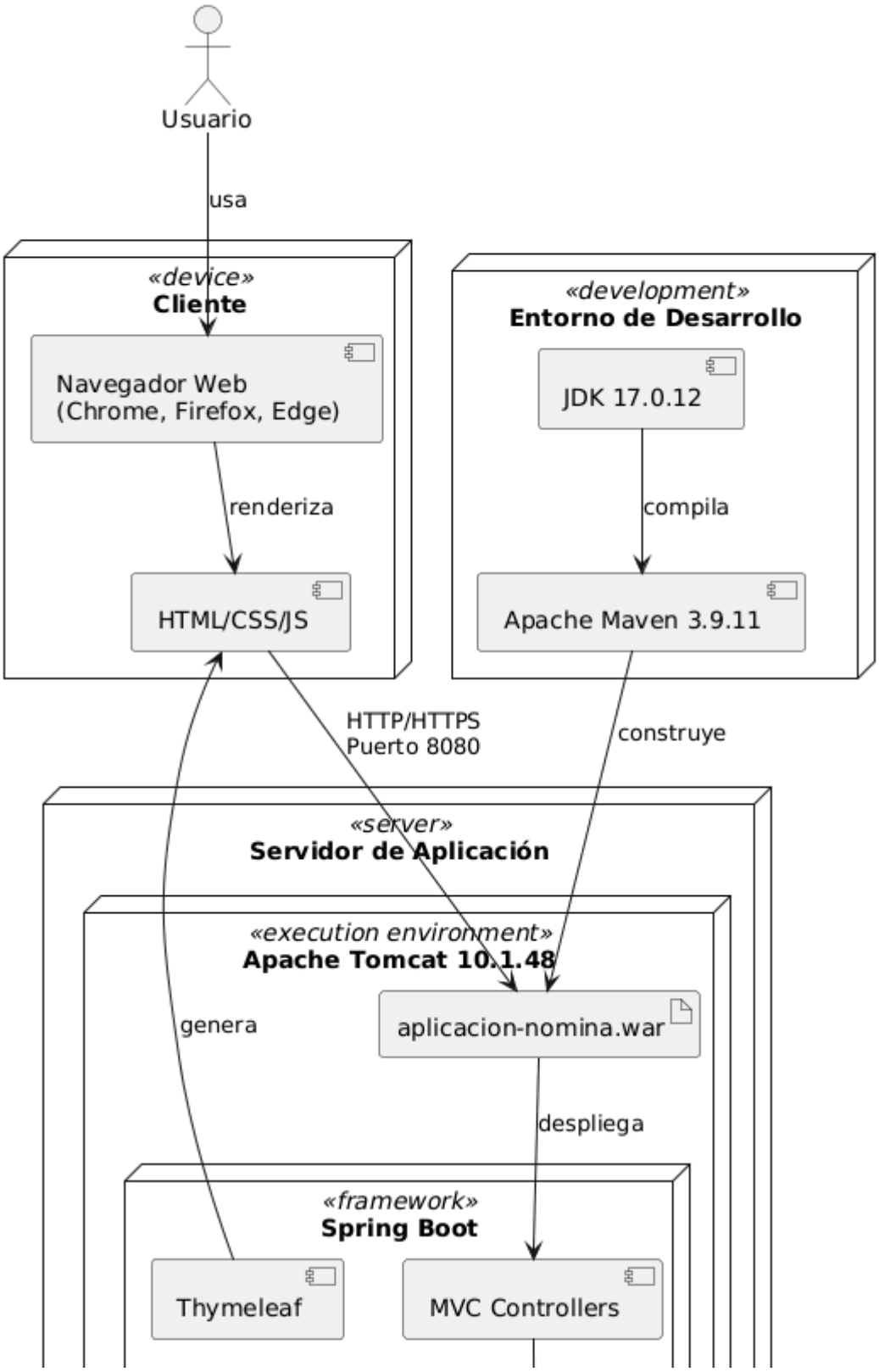
- **Capa de presentación.** Se encargará de manejar las vistas, i. e., será la GUI.
  - **MVC**
    - Bajo este paradigma se maneja la comunicación y despliegue de las vistas a través de los componentes llamados controladores.
  - **View**
    - Interfaces codificadas en html, con su respectivos estilos en CSS y funcionalidades básicas en Java Script.
  - **Controller**
    - Serie de componentes que manejan las vistas.
- **Capa de negocio.** Capa donde se llevan a cabo operaciones y transacciones para el funcionamiento esperado del sistema en base a las peticiones hechas en la capa de presentación.
  - **Services**
    - Encargados de procesar las peticiones hechas desde las vistas y entregadas a través del controlador, a este último se le brindará la respuesta a la petición.
  - **Domain**
    - Define la estructura de la base de datos y contendrá elementos simples para el manejo de operaciones.
  - **Util**
    - Contendrá operaciones, operaciones de validación, así como funcionalidades generales.
- **Capa de acceso a datos.** Proporciona la comunicación entre la capa del negocio y la base de datos.
  - **Repository**
    - Se encarga de brindar desde Spring boot la posibilidad de hacer operaciones del CRUD en la base de datos. Es solicitado por la capa de lógica del negocio para acceder a información puntual.
- **Base de datos**
  - Base de datos en H2

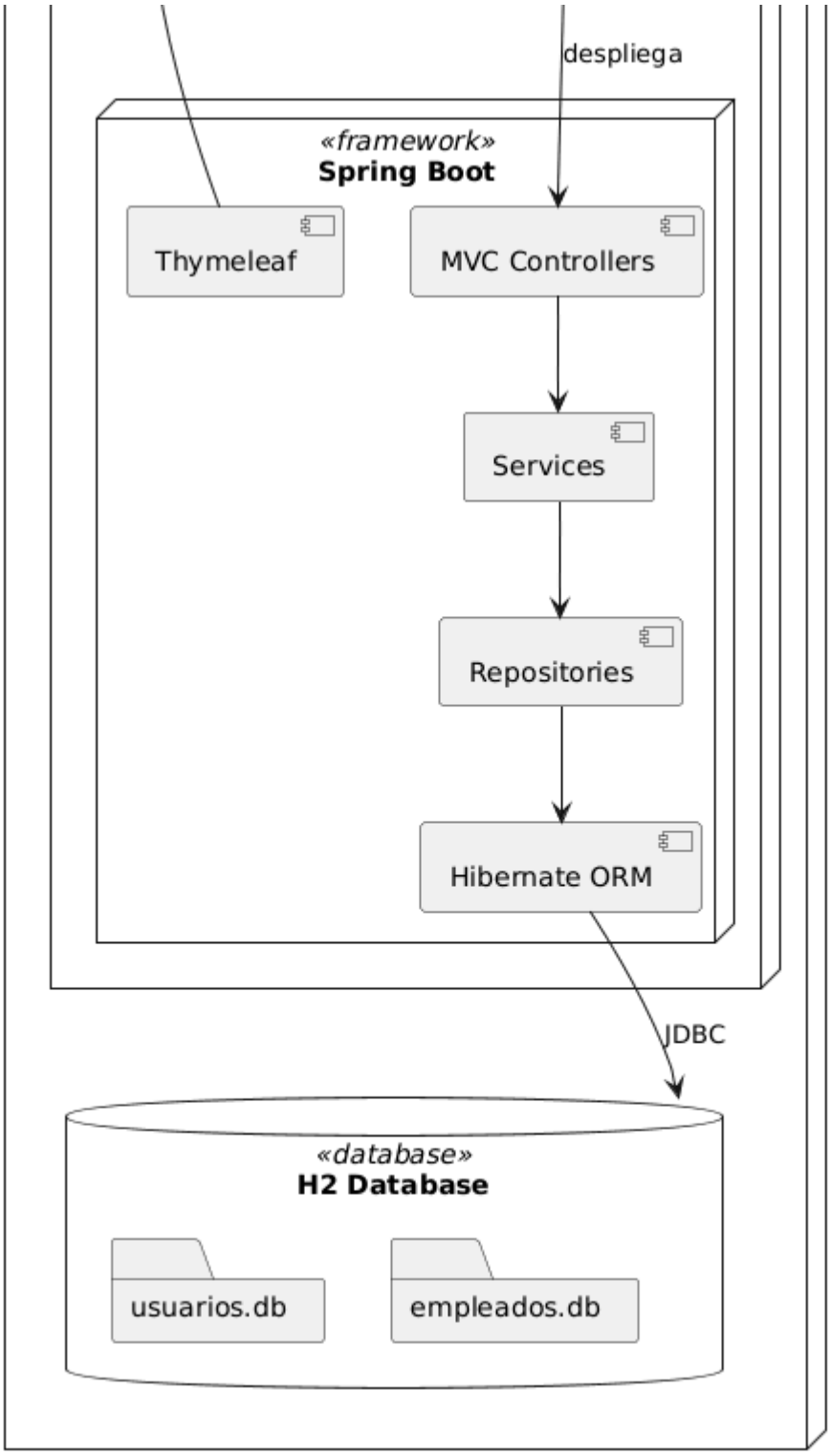
Se profundizará en este esquema en la fase de construcción para incluir aspectos específicos de la construcción y manejo del sistema.

2.6. Diagrama de componentes



2.7. Diagrama de despliegue



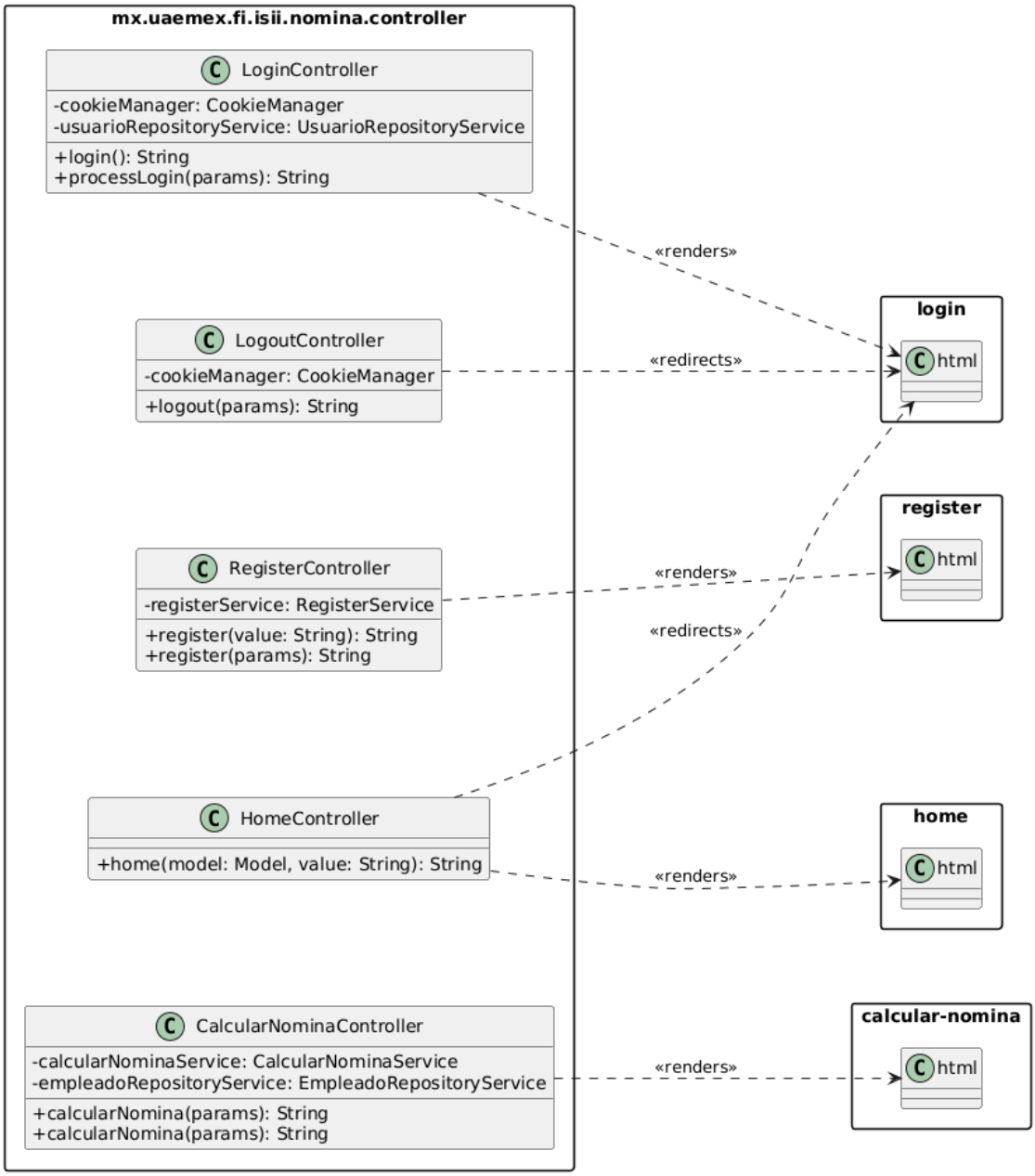

despliega
«framework»
Spring Boot
Thymeleaf
MVC Controllers
Services
Repositories
Hibernate ORM
JDBC
«database»
H2 Database
usuarios.db
empleados.db

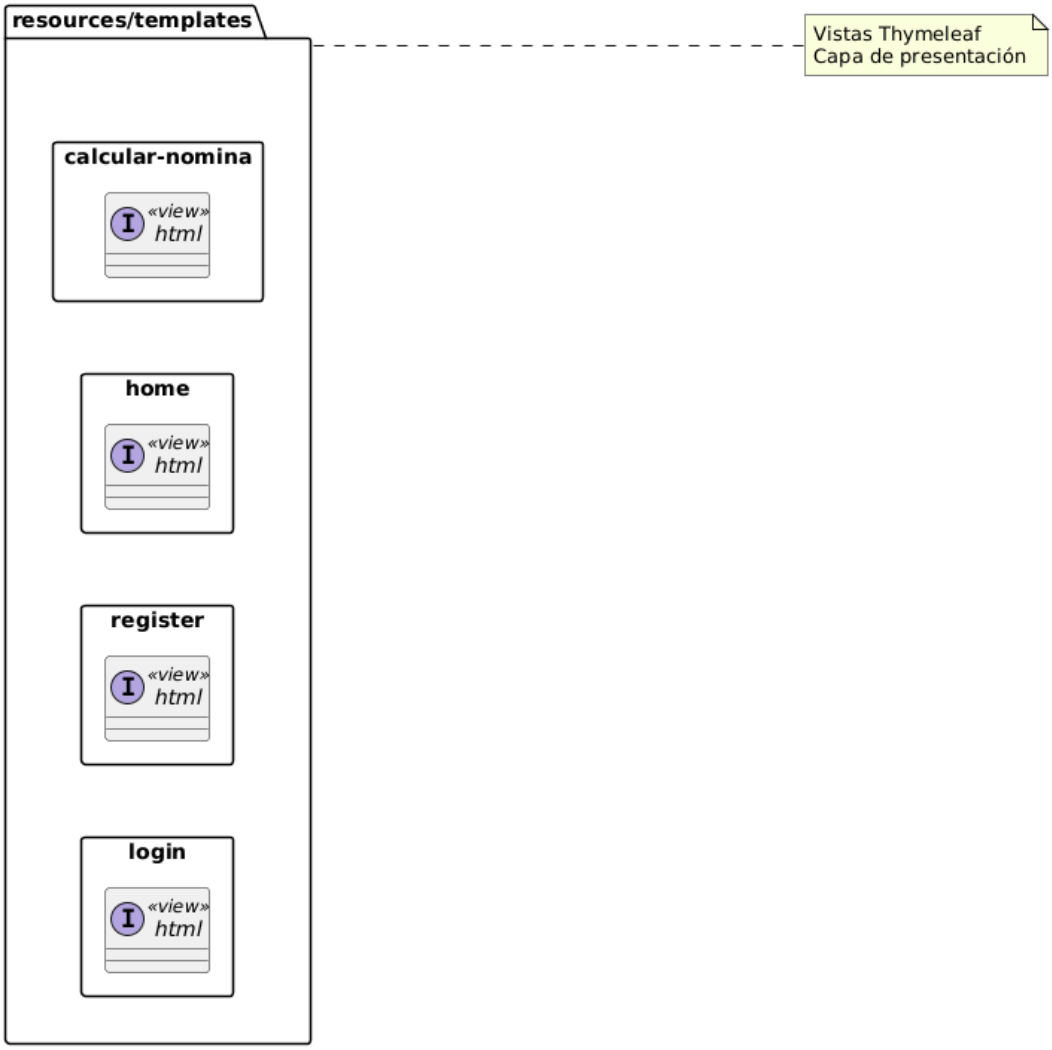
3. Construcción

3.1. Diagramas de clases

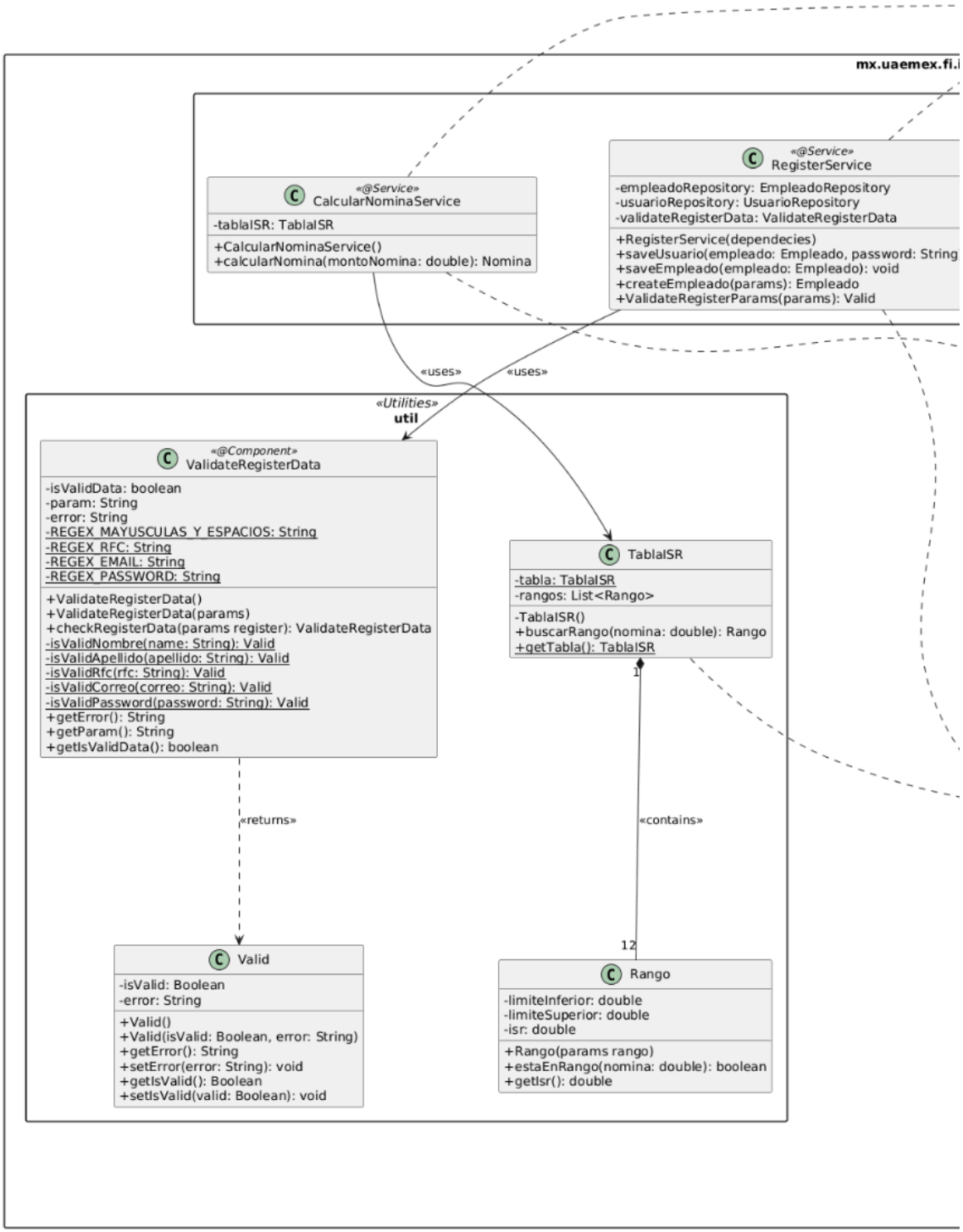
Para simplificar el contenido de cada diagrama, solo se incluyen aquellos métodos y funcionalidades fundamentales de cada clase.

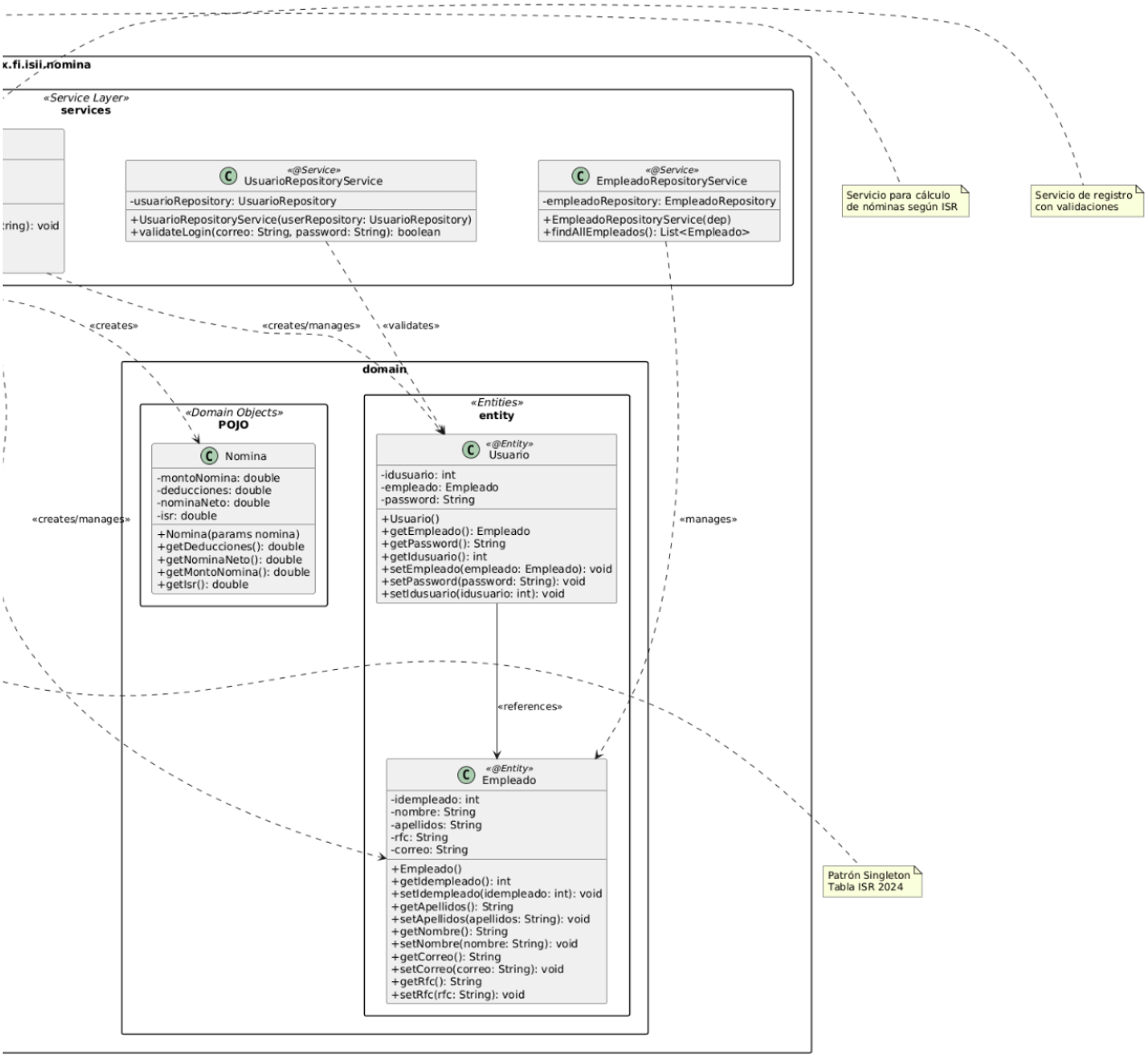
3.1.1. Capa de presentación




resources/templates
Vistas Thymeleaf
Capa de presentación
calcular-nomina
«view»
html
home
«view»
html
register
«view»
html
login
«view»
html

3.1.2. Capa del negocio





El paquete **services** proporciona las operaciones fundamentales de la capa de negocios, y la comunica con la capa de presentación, así mismo, recibe, envía y procesa peticiones o en su defecto datos, es la parte medular de esta capa.

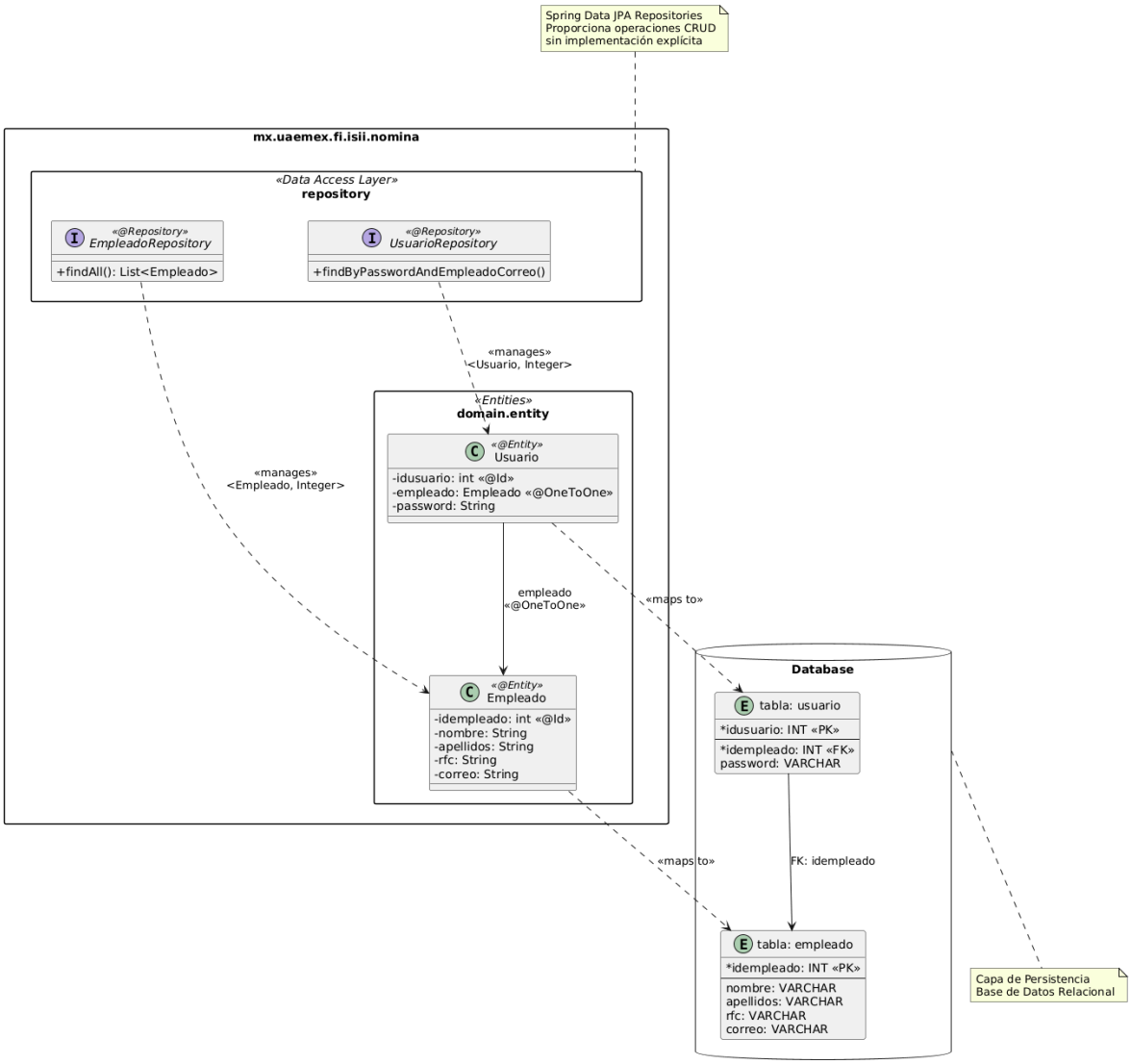
Services

- **CalcularNominaService**. Realiza el cálculo de la nómina, para ello, recibe del controlador únicamente el monto del salario, para hacer el cálculo se apoya del paquete utils el cual a través de las clases **utils/TablaISR** y **utils/Rango**, permite obtener los datos necesarios para hacer el cálculo de la nómina, al final

retorna el salario, el isr correspondiente a ese salario, las deducciones, y el salario neto.

- **RegisterService**. Recibe del controlador los parámetros necesarios para realizar el alta de un empleado o usuario según sea el caso, a través de la clase **utils/ValidateRegisterData** valida cada uno de los parámetros y maneja casos en que alguno de los parámetros no venga en el formato adecuado, auxiliándose de la clase utils/Valid para manejar las validaciones de cada campo. Si las validaciones son correctas, a través de **EmpleadoRepositoryService** y **UsuarioRepositoryService** realiza las peticiones para el alta del registro.
- **EmpleadoRepositoryService y UsuarioRepositoryService:** Clases utilizadas para comunicar la capa de acceso a datos con la capa del negocio, brindan a **RegisterService** el medio de comunicación para solicitar el alta de un registro en la BD.

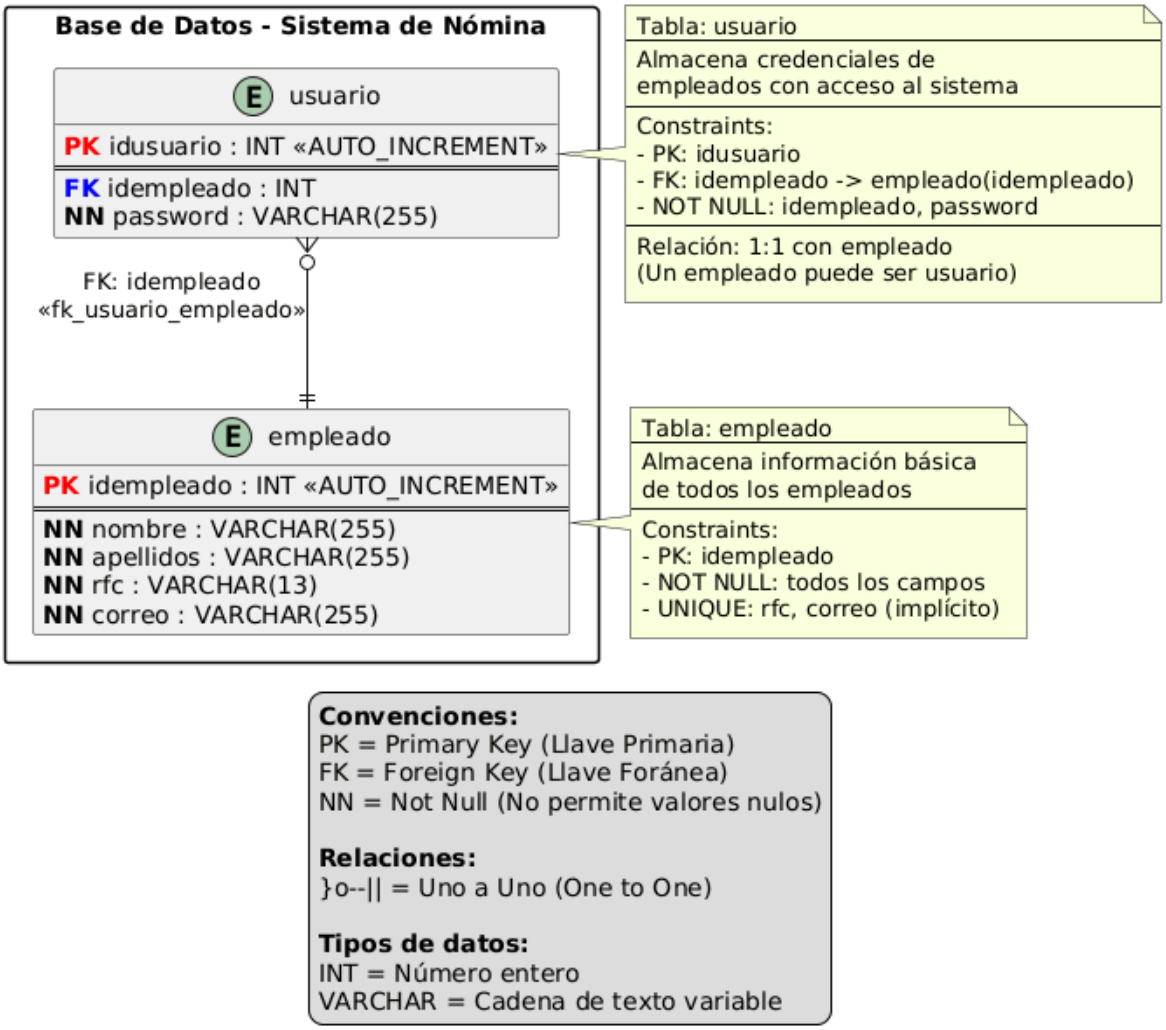
3.1.3. Capa de acceso a datos



Repository

Brinda a través de las clases **EmpleadoRepository** y **UsuarioRepository** la facultad de hacer operaciones de CRUD y acceder a la BD, la cual fue definida en el esquema de las tablas definidas en **domain/entity/.**

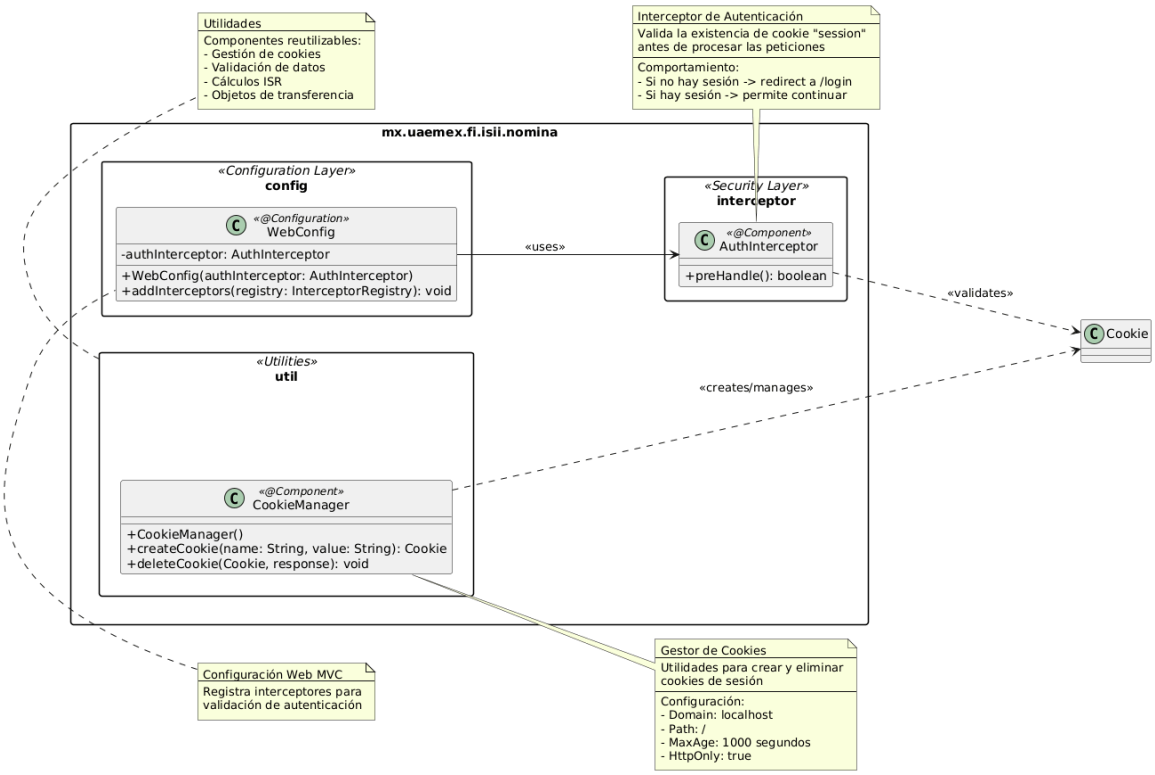
3.1.4. Capa de Base de datos



El esquema de la base de datos está definido en **domain/entity**.

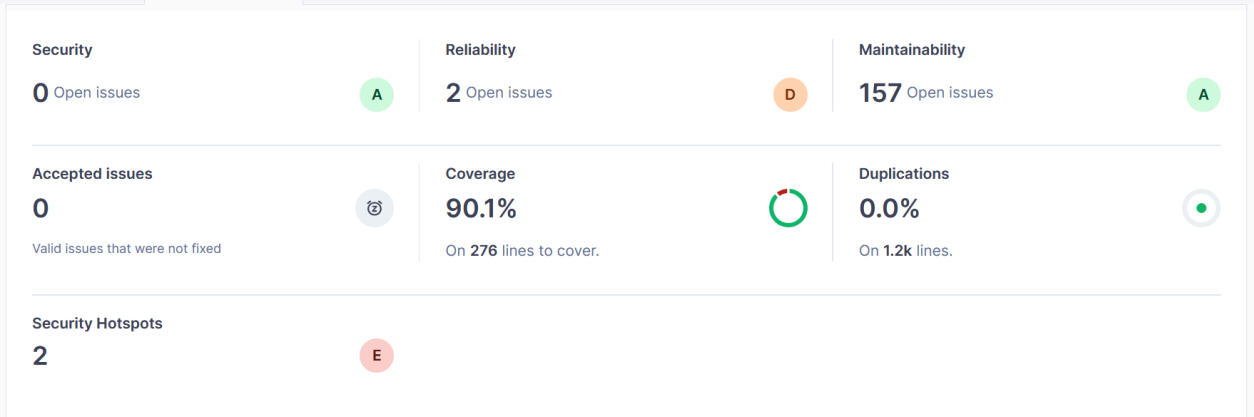
3.1.5. Capa independiente

En esta capa, se consideran aquellos componentes que fungen propósitos específicos de la configuración, seguridad y manejo de la aplicación.



4. Pruebas

SonarCube Análisis



Pruebas unitarias y de integración

Las pruebas se aplicaron en aquellas **clases y métodos críticos** para el programa. Getters, setters y constructores fueron omitidos. Quedan fuera del alcance de estas pruebas los paquetes domain, repository, interceptor y config.

.utils

ValidateRegisterData

ID	Método	Descripción	Entrada	Salida esperada	Estado
UT-VRD-001	isValidName	Campo nulo	null	FALSE	
UT-VRD-002		Campo demasiado corto	a	FALSE	
UT-VRD-003		Todo en minúsculas	jose	FALSE	
UT-VRD-004		Formato correcto	JOSE	TRUE	
UT-VRD-005	isValidApellido	Campo nulo	null	FALSE	
UT-VRD-006		Campo demasiado corto	j	FALSE	
UT-VRD-007		Todo en minúsculas	Jimenez	FALSE	
UT-VRD-008		Formato correcto	JIMENEZ	TRUE	
UT-VRD-009	isValidRfc	Campo nulo	null	FALSE	
UT-VRD-010		Longitud no igual a 13	ASVS899898	FALSE	
UT-VRD-011		Formato incorrecto	EJEMPLORFC122	FALSE	
UT-VRD-012		Formato correcto	EJEM110011PLO	TRUE	
UT-VRD-013	isValidCorreo	Campo nulo	null	FALSE	
UT-VRD-014		Campo demasiado corto	a@a.a	FALSE	
UT-VRD-015		Formato incorrecto	preuba@leng	FALSE	

UT-VRD-016		Formato correcto	prueba@lenght.ok	TRUE	
UT-VRD-017	isValidPassword	Campo nulo	null	FALSE	
UT-VRD-018		Longitud menor a 8	pAs11%m	FALSE	
UT-VRD-019		Formato incorrecto	contrasena123	FALSE	
UT-VRD-020		Formato correcto	aB1$cD2E	TRUE	
UT-VRD-021	isValidPassword	Sin mayúscula	password123!	FALSE	
UT-VRD-022		Sin minúscula	PASSWORD123!	FALSE	
UT-VRD-023		Sin número	Password!	FALSE	
UT-VRD-024		Sin carácter especial	Password123	FALSE	
UT-VRD-025	isValidNombre	Contiene un número	JUAN123	FALSE	
UT-VRD-026	IsValidApellido	Contiene un carácter especial	PEREZ@GARCIA	FALSE	
UT-CRD-001	checkRegisterData	Registro completo empleado	Todos los campos del empleado llenos	isValidData: true, error:	
UT-CRD-002		Registro completo admin	Todos los campos del admin llenos	isValidData: true, error: ""	
UT-CRD-003		Todos campos nulos	Todos campos nulos	isValidData: false, error: "'Nombre' no puede estar vacio"	
UT-CRD-004		Múltiples errores	nombre: null, apellido: null, rfc: "ABC", correo: "invalid", isAdmin: true, password: "weak"	isValidData: false, error: "'Nombre' no puede estar vacio"	
UT-CRD-005		isAdmin null sin password	nombre: "JUAN", apellido: "PEREZ",	isValidData: true	

			rfc: "ABCD123456XYZ", correo: "test@example.com", isAdmin: null, password: null		
UT-CRD-006		isAdmin false null password	nombre: "JUAN", apellido: "PEREZ", rfc: "ABCD123456XYZ", correo: "test@example.com", isAdmin: null, password: null	isValidData: true	
UT-CRD-007		Admin sin password	nombre: "JUAN", apellido: "PEREZ", rfc: "ABCD123456XYZ", correo: "test@example.com", isAdmin: true, password: null	isValidData: false, error: "'Password' no puede estar vacio"	
UT-CRD-008		Validar parámetro retornado	nombre: "JUAN", apellido: "PEREZ", rfc: "INVALID", correo: "test@example.com", isAdmin: false, password: null	isValidData: false, param: "INVALID", error: "Longitud de RFC demasiado corta"	
UT-CRD-009		Correo institucional	nombre: "JUAN", apellido: "PEREZ", rfc: "ABCD123456XYZ", correo: "juan.perez@uaemex.mx", isAdmin: null, password: null	isValidData: true	
UT-CRD-010		Objeto retornado válido	nombre: "JUAN", apellido: "PEREZ", rfc: "ABCD123456XYZ", correo: "test@example.com", isAdmin: null, password: null	isValidData: true, error: "", param: ""	

TablaIsr

ID	Método	Descripción	Entrada	Salida esperada	Estado
UT-TISR-001	buscarRango	Nómina en límite inferior del Rango 1	nomina = 0	isrEsperado = 0.0	
UT-TISR-002		Nómina en límite inferior del Rango 2	nomina = 8364.1	isrEsperado = 1.92	
UT-TISR-003		Nómina en límite inferior del Rango 3	nomina = 8952.50	isrEsperado = 6.40	
UT-TISR-004		Nómina en límite superior del Rango 1	nomina = 8364.1	isrEsperado = 0.0	
UT-TISR-005		Nómina en límite superior del Rango 2	nomina = 8952.49	isrEsperado = 1.92	
UT-TISR-006		Nómina en rango 1	nomina = 5000.0	isrEsperado = 0.0	
UT-TISR-007		Nómina en rango 3	nomina = 50000.0	isrEsperado = 6.40	
UT-TISR-008		Nómina en rango 4	nomina = 100000.0	isrEsperado = 10.88	
UT-TISR-009		Nómina en rango 9	nomina = 800000.0	isrEsperado = 30.00	
UT-TISR-010		Nómina en el último rango	nomina = 5000000.0	isrEsperado = 35.00	
UT-TISR-011		Verificar funcionamiento de singleton	TablaISR tabla1 = TablaISR.getTabla(); TablaISR tabla2 = TablaISR.getTabla();	tala 1 = tabla 2	

Dado que la construcción de los rangos de la tabla se hizo de manera uniforme y análoga en cada caso, se asume que si al menos 3 casos son correctos, los demás lo serán.

Rango

El método estaEnRango es el único método con lógica crítica.
Existe una fuerte dependencia del método estaEnRango() con la clase TablaISR.

ID	Método	Descripción	Entrada	Salida esperada	Estado
UT-R-001	estaEnRango	Valor de nómina muy alto	10000000.0	TRUE isr = 35.0	
UT-R-002		Nómina diferente para primer y segundo rango	nomina = 8364.0 nomina = 8364.01	primerRango = TRUE nomina2 -> primerRango = FALSE	
UT-R-003		Nómina diferente para segundo y tercer rango	nomina = 8952.49 nomina = 8952.5	segundoRango = TRUE nomina2 -> tercerRango = FALSE	
UT-R-004		Nómina valor medio	nomina = 100000.0	TRUE isrEsperado = 10.88	
UT-R-005		Valor negativo de nómina	nomina = -1	FALSE	
UT-R-006		Valor típico para la nomina.	nomina = 15000.0	TRUE isrEsperado = 6.40	

CookieManager

ID	Método	Descripción	Entrada	Salida esperada	Estado
UT-CM-001	createCookie	Parámetros válidos	name = "sessionId"; value = "abc123";	not null	
UT-CM-002		El path de la cookies es /	name = "sessionId"; value = "abc123";	“/” = cookie.getPath()	
UT-CM-003	deleteCookie	Establece el tiempo de vida de la cookie en 0	cookie	TRUE	
UT-CM-004	createCookie, deleteCookie	Verifica que la cookie fue creada y eliminada	Parametros para creación de una cookie y despues esta cookie	TRUE	

.services

UsuarioRepositoryService

ID	Método	Descripción	Entrada	Salida esperada	E.
UT-URS-001	validateLogin	Los datos son correctos, y existen en la BD	correo = "usuario@example.com"password = "password123";	TRUE	
UT-USR-002		Password no válido	Simulación de un correo válido, y una password inválida	FALSE	
UT-USR-003		Correo no válido	Simulación de un correo inválido, y una password válida	FALSE	
UT-USR-004		Password y correo no existen	Parámetros de prueba	FALSE	
UT-USR-005		Intento de inyección SQL	Simulación de inyección SQL a través de los campos password y correo	FALSE	

EmpleadoRepositoryService

ID	Método	Descripción	Entrada	Salida esperada	E.
UT-ERS-001	findAllEmpleados	Encontrar a todos los empleados, la BD está vacía	Solicitud de todos los registros	Lista vacía TRUE	
UT-ESR-002		Encontrar a todos los empleados, solo hay un registro	Solicitud de todos los registros	Lista con un empleado	
UT-ESR-003		Encontrar a todos los empleados, hay un número arbitrario de registros	Solicitud de todos los registros	Lista con todos los empleados	

RegisterService

ID	Método	Descripción	Entrada	Salida esperada	E.
UT-RS-001	saveUsuario	Guarda correctamente un empleado de tipo usuario	Datos de un usuario	Save successful	
UT-RS-002		Verifica que el usuario tenga asociado el empleado correcto	Registro de un usuario. Id del empleado	Datos coincidentes	
UT-RS-003	saveEmeplado	Guarda correctamente un empleado	Datos de un empleado	Save successful	
UT-RS-004	createEmpleado	Crea un empleado con datos válidos	Datos de un empleado	Datos coincidentes	
UT-RS-005	validateRegisterParams	Todos los parámetros válidos	Parámetros válidos	isValidData = true	
UT-RS-006		Nombre invalido	Parámetros válidos, excepto el nombre	isValidData = false	
UT-RS-007		Apellido invalido.	Parámetros válidos, excepto el apellido	isValidData = false	
UT-RS-008		Rfc invalido	Parámetros válidos, excepto el rfc	isValidData = false	
UT-RS-009		Correo invalido	Parámetros válidos, excepto el correo	isValidData = false	
UT-RS-010		Empleado de tipo usuario. Password invalido.	Parámetros válidos, excepto el password	isValidData = false	
UT-RS-011	createEmpleado, saveEmpleado	Crear y guardar empleado	Empleado válido	Save successful	

UT-RS-012	createEmpleado, saveUsuario	Crear empleado y guardar usuario	Usuario valido	Save successful	
UT-RS-013	validateRegisterData, createEmpleado, saveEmpleado, saveUsuario	Flujo completo de un registro de usuario	Parámetros válidos, empleado válido, usuario válido	Save successful	
UT-RS-014	saveUsuario	Retorno de excepción al guardar un usuario	Excepción en la BD	catch Exception	
UT-RS-015	saveEmpleado	Retorno de excepción al guardar un usuario	Excepción en la BD	catch Exception	
UT-RS-016	validateRegisterData	Múltiples campos inválidos	Múltiples campos de empleado y usuario inválidos	isValidData = false	

CalcularNominaService

Lógica fundamental ya manejada en utils/TablaISR y utils/Rango.

ID	Método	Descripción	Entrada	Salida esperada	E.
UT-CNS-001	calcularNomina	Utiliza correctamente los rangos de la tabla de isr	Diversos montos de nómina	Los valores del ISR son iguales a los esperados	
UT-CNS-002	Constructor	La tabla de isr se inicializa correctamente	Invocación al constructor	TablaIsr != null	

.controller

LoginController

ID	Método	Descripción	Entrada	Salida esperada	E.
UT-LC-001	GET /login	Verifica que se retorna la vista del login	Método GET para /login	TRUE	
UT-LC-002	POST /login	Credenciales válidas, redirige a /home	POST con parámetros para /login	redirect:/home	
UT-LC-003		Credenciales válidas, crea cookie	POST con parámetros para /login	not null createdCookie	
UT-LC-004		Credenciales inválidas. Retorna /login	POST con parámetros para /login	/login	
UT-LC-005		Se llama a validateLogin del service	POST con parámetros para /login	validateLogin fue ejecutado	
UT-LC-006		Flujo correcto del login	POST con parámetros para /login. Se llaman a todos los métodos involucrados	validateLogin se ejecuto, se creó la cookie, hubo un redireccionamiento y consulta en la BD	
UT-LC-007		FLujo correcto del login, en caso de credenciales inválidas	POST con parámetros para /login. Se llaman a todos los métodos involucrados	validateLogin se ejecuto,no se creó la cookie, se hizo una consulta en la BD, no hubo resultado.	

HomeController

UT-HC-001	GET /home	Retorna la vista si la cookie de sesión existe	GET /home Cookie	/home	
UT-HC-002	POST /home	Acceso a /home sin cookie válida.	GET /home	redirect:/login	
UT-HC-003		Flujo correcto. ValidateLogin creó una cookie de sesión y se accede a /home	POST /ogin con parámetros credenciales válidas, sessionCookieValue	redirect:/home	
UT-HC-004	POST /home	Sesión nula. No puede acceder a /home	nullSession	redirect:/login	

CalcularNominaController

ID	Método	Descripción	Entrada	Salida esperada	E.
UT-CNC-001	GET /calcular-Nomina	Verifica que se retorna la dirección /calcular-nomina	Método GET para /calcular-nomina	TRUE	
UT-CNC-002		Verifica que se despliegue una lista de empleados	Método GET para /calcular-nomina	Lista de empleados	
UT-CNC-003		La lista de empleados está vacía	Método GET para /calcular-nomina	empleadosVacios.isEmpty()	
UT-CNC-004	POST /calcular-nomina	Se agregan todos los atributos al modelo	Atributos de model	Ningún atributo está vacío y pueden ser accedidos desde el model	
UT-CNC-005	POST /calcular-nomina	Retorna la vista correcta	Método POST para /calcular-nomina	TRUE	
UT-CNC-006		Maneja la excepción en caso de que el calculo de nomina falle	Monto de nómina negativo, causa excepción	redirect:/calcular-nomina Resultado en caso de excepción	

RegisterController

ID	Método	Descripción	Entrada	Salida esperada	E.
UT-RC-001	GET /register	Recibe metodo GET en /register, con cookie de session	Método GET para /register Cookie	TRUE	
UT-RC-002		Recibe metodo GET en /register, sin cookie de session	Método GET para /register	redirect:/login	
UT-RC-003	POST /login	Se envían los parámetros, se crea un empleado	POST con parámetros para /register en caso de un empleado	Mensaje de éxito != null	
UT-RC-004	POST /login	Se envían los parámetros, se crea un empleado y un usuario	POST con parámetros para /register en caso de un usuario	Mensaje de éxito != null	

UT-RC-005		Falla la validación de los datos. Agrega el error al la pagina	POST con parámetros para /register	Mensaje de error != null	
UT-RC-006		Falla la validación de los datos. Agrega el error al la pagina	POST con parámetros para /register	No se ejecuta el metodo createEmpleado	
UT-RC-007		Excepcion al guardar el empleado	POST con parámetros para /register	Mensaje de error != null redirect:/register	

LogoutController

ID	Método	Descripción	Entrada	Salida esperada	E.
UT-LGC-001	GET /logout	Recibe metodo GET en /logout, con cookie de session. Elimina la cookie	Método GET para /logout Cookie	La cookie es eliminada	
UT-LGC-002		Recibe metodo GET en /logout, con cookie de session. Redirige a /	Método GET para /logout Cookie	redirect:/	
UT-LGC-003		Recibe metodo GET en /logout, con cookie de session. Llama a delete cookie al menos una vez	Método GET para /logout Cookie	deleteCookie es llamado	
UT-LGC-004		Recibe metodo GET en /logout, sin cookie de session. Redirige a /	Método GET para /logout	redirect:/	
UT-LGC-005		Revisar el flujo completo del logout. Primero el login, acceso a home, luego a /logout	Flujo completo del logout. Primero el login, acceso a home, luego a /logout	Los metodos participes en estos procesos se ejecutan cuando es necesario, se crean y eliminan las cookies	

5. Diccionario de términos

- **Nómina**: Suma de los registros financieros de los sueldos de los empleados, incluyendo los salarios, el ISR correspondiente al salario, las deducciones y el salario neto.
- **Usuario**: Aquella persona que poseerá credenciales de acceso al sistema.
- **Tabla de deducciones**: Tabla que de acuerdo a los documentos oficiales del SAT, permitirá obtener el ISR correspondiente al monto salarial.
- **ISR:** Impuesto sobre la renta. Impuesto que se aplica al salario.
- **Cookie**: Pequeño segmento de almacenamiento local para manejar información de la sesión del usuario.
- **Prueba unitaria**: Prueba que se lleva a cabo sobre una sección o un método particular del código.
- **Prueba de integración**: Prueba que se lleva a cabo para determinar si la comunicación e interacción entre las distintas secciones de código se lleva a cabo de manera eficaz.
- **Frontend**: Interfaz de usuario.
- **Backend**: Sistema y lógica del negocio, que se encarga de brindar el funcionamiento al frontend y realizar operaciones y transacciones.
- **Arquitectura**: En el software, se refiere a cómo los componentes de un sistema informático estarán estructurados y cómo será su comportamiento.
- **Requerimientos funcionales**: Aquellas funcionalidades o características que el usuario esperaría del sistema.
- **Requerimientos no funcionales:** Se refiere a todas características que el usuario no tiene en cuenta, pero deben considerarse en la construcción del sistema, para su correcto funcionamiento.
- **GUI**: Interfaz del usuario.

6. Referencias

- Servicio de Administración Tributaria. (2024, 30 de diciembre). *Anexo 8 de la Resolución Miscelánea Fiscal para 2025*. https://www.sat.gob.mx/minisitio/NormatividadRMFyRGCE/documentos2025/rmf/anexos/Anexo8_RMF2025-30122024.pdf
- Servicio de Administración Tributaria (SAT). (s.f.). *Estructura de la clave de Registro Federal de Contribuyentes (RFC)*. https://ampocdevbuk01a.s3.us-east-1.amazonaws.com/Estructura_de_la_clave_de_Registro_Federal_de_Contribuyentes_RFC_5ab4a474bb.pdf