UNIVERSIDAD AUTONOMA DE YUCATAN

ARQUITECTURAS DE SOFTWARE (AS-G1)

PROYECTO FINAL
CRUD EN UN FRAMEWORK

PROFESOR:
VICTOR HUGO MENENDEZ DOMINGUEZ

INTEGRANTES DE EQUIPO:
-ALEXIS ROSALDO
-AMAURY MORALES

FECHA DE ENTREGA: 6 DE DICIEMBRE 2021

Contenidos

Bitácora de cambios del documento

Versión	Fecha	Autores	Razón
1	2/12/2021	Alexis Amaury	Estructura del documento inicial y llenado de apartados
2	3/12/2021	Alexis Amaury	Finalización del documento

Definiciones y abreviaciones

Del proyecto

#	Palabra	Definición
1	CRUD	Acrónimo del inglés “Create, Read, Update, Delete”. Es la descripción de las funcionalidades de un sistema de altas y de bajas con habilidades de modificación y consulta.
2	Framework	Del inglés “Marco de trabajo”. Es una estructura conceptual y tecnológica de asistencia definida, normalmente, con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.
3	Front-End	El Front-End es la capa que se dedica a la parte frontal de, por ejemplo, un sitio web. La estructura del sitio, los estilos, el diseño y la interacción directa con el usuario final es parte del Front-End
4	Back-End	El Back-End es la capa que se dedica a la parte trasera de, por ejemplo, un sitio web. Normalmente está oculta fuera del alcance de los usuarios. Acceso a los datos y modificaciones a los mismos son parte del Back-End
5	UML	Acrónimo del inglés “Unified Modeling Language”. Es un lenguaje de modelado visual común y semántica y sintácticamente rico para la arquitectura, el diseño y la implementación de sistemas de software complejos

De las tecnologías

#	Palabra	Definición
1	HTML	Acrónimo del inglés “HyperText Markup Language”, un sistema estandarizado para etiquetar archivos de texto para obtener fuentes, colores, gráficos, y enlaces en las páginas web.
2	SQL	Acrónimo del inglés “Structured Query Language”. Es un lenguaje estructurado para interactuar con bases de datos.
3	MySQL	Es un sistema administrador de bases de datos tipo SQL
4	NodeJS	Es un entorno de ejecución de JavaScript orientado a eventos asíncronos, diseñado para crear aplicaciones network escalables.
5	ExpressJS	Es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles.
6	Javascript, JS	Es un lenguaje de programación ligero, interpretado, generalmente utilizado en el desarrollo de páginas web.
7	JQuery	Es una librería pequeña y rápida de Javascript que permite controlar los elementos de una página web en tiempo real después de haber cargado la página, entre otras cosas.
8	Bootstrap	Es un framework para construir sitios web responsivos en minutos, que sean amigables para interfaces de dispositivos móviles.
9	Font Awesome	Un conjunto de herramientas, fuentes e iconos.
10	Spring	Es un framework para el desarrollo de aplicaciones en Java, comúnmente utilizada para facilitar y flexibilizar la creación de objetos en el programa.
11	HTTP	Acrónimo del inglés “HyperText Transfer Protocol” Es el nombre de un protocolo el cual permite realizar una petición de datos y recursos, como un archivo HTML.
12	GET POST PUT DELETE	Verbos HTTP, métodos de petición de recursos. GET: Del inglés “obtener” POST: Del inglés “enviar” PUT: Del inglés “poner” DELETE: Del inglés “eliminar”
13	REST	Acrónimo del inglés ”REpresentational State Transfer”

		Es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP
14	API	Es una forma de describir la forma en que los programas o los sitios webs intercambian datos.

Descripción General del Proyecto

Objetivo

Este es el proyecto final de la asignatura de Arquitecturas de Software, el cual es la creación de una aplicación CRUD utilizando marcos de trabajo presentados en la actividad “Marcos de trabajo” del curso.

A grandes rasgos, la aplicación le permite al usuario:

- Agregar nuevos registros a la base de datos
- Modificar registros ya existentes en la base de datos
- Eliminar registros ya existentes en la base de datos
- Consultar registros existentes en la base de datos con una interfaz gráfica

Alcance

La aplicación debe cumplir solo con todo lo siguiente:

La aplicación debe mostrar el listado de registros de una tabla de una base de datos MySQL.
La tabla debe de tener al menos 3 campos:

- 1 identificador único
- 1 campo de texto
- 1 campo numérico

Cada registro listado en la tabla tendrá una casilla de verificación para indicar que es el registro seleccionado.
Por cada registro, el usuario podrá editar sus valores o eliminarlo.
En la parte superior de la tabla deben aparecer botones para crear un registro o borrar los registros seleccionados.

En un futuro se puede extender para incluir otro tipo de funcionalidades, como búsqueda, paginación, entre otros.

Descripción de Funcionalidades

Base de datos

Módulo	Base de Datos
Objetivo Principal	Este módulo se encarga de la configuración, conexión y acceso a la base de datos MySQL creada utilizando phpMyAdmin
Funcionalidades	● Almacena información ● Manipula la información almacenada con respecto a las sentencias SQL recibidas.

Administrador de Peticiones

Módulo	Administrador de Peticiones HTTP de spring y configuración y acceso a la base de datos
Objetivo Principal	Manipula las peticiones recibidas en formato REST API por parte del front end y configura los conectores y sentencias SQL para manipular información de la base de datos.
Funcionalidades	Compuesta de los paquetes: **Paquete:** com.employees.spring **Clase:** AppConfiguration.java **Clase:** SpringEmployesApplication.java **Paquete:** com.employees.spring.controllers **Clase:** EmployeeController.java **Paquete:** com.employees.spring.daos **Clase:** <<Interface>> MysqlConnector.java **Clase:** MysqlConnectorImpl.java **Paquete:** com.employees.spring.entities **Clase:** Employee.java ● Rest Controllers: Se encarga de recibir las peticiones HTTP que se reciben en formato JSON (de acuerdo al protocolo REST API) y efectuar cada uno de los verbos correspondientes dependiendo del tipo de solicitud.

	• DAO con Mysql usando JDBC: Usando las librerías integradas que ofrece el entorno de trabajo spring relacionadas al conector JDBC e implementando clases e interfaces de tipo DAO, se conectan a la base de datos y manejan su contenido mediante sentencias SQL. • DataBase Config: Se encarga de definir los parámetros mediantes los cuales se tiene que acceder a la base de datos y poder establecer la conexión y la manera de definir las sentencias SQL.

Servidor Web

Módulo	Servidor web
Objetivo Principal	Configura el servidor web que sirve la página web a partir de su enrutamiento.
Funcionalidades	Compuesto por los siguientes elementos: **Archivo:** index.js Donde se configura el servidor en NodeJS y se hace la configuración de enrutamiento con ExpressJS. • Configura y pone en marcha un servidor web en un puerto cerrado dentro del localhost • Enruta los URL para poder servir la página web dependiendo de la dirección ingresada

Interfaz Grafica (Pagina Web)

Módulo	Interfaz gráfica (Página web)
Objetivo Principal	Muestra un panel de control para manipular la información que se recibe de la base de datos MySQL a partir de una serie de peticiones realizadas tipo REST API
Funcionalidades	Compuesto por los siguientes elementos: **Archivo:** crud.js La parte lógica de la interfaz, donde se manejan los eventos de botones y se realizan las operaciones en la tabla. También incluye las peticiones REST API. **Archivo:** index.html El archivo HTML principal de la página web.

	● Muestra todos los registros localizados dentro de la base de datos. ● Por cada registro, el usuario puede editar sus valores o eliminarlo. ● El usuario puede crear nuevos registros. ● El usuario puede eliminar varios registros a la vez seleccionando sus casillas correspondientes o con la casilla para seleccionar todos los registros. ● Valida la información introducida por el usuario.

Descripción de Componentes

Front End

Componente	Front End
Descripción	Se encarga de mostrar una interfaz de usuario que manipula los datos suministrados por el Front End y envía peticiones de cambios al Back End.
Dependencias con otros componentes	REST API Back End para poder comunicar las peticiones directamente con el Back-End y recibir sus respuestas
Interfaces de Salida	Interfaz Gráfica representando una tabla que contiene los registros almacenados en la base de datos, junto con botones que pueden modificar dichos datos, eliminarlos, o agregar más.
Interfaces de Entrada	La información nueva que nos da el usuario se transforma en nuevos registros o modificaciones a los mismos. En este caso: Formulario de Empleado ● name: String. el nombre del empleado ● email: String. el correo del empleado ● address: String. la dirección del empleado ● phone: Integer. el teléfono del

	empleado ● id: Integer. el identificador del empleado, suministrado automáticamente por el programa.
Artefactos	NodeJS, ExpressJS: Para poner en marcha el sitio web en el puerto 8000 del localhost. Bootstrap, FontAwesome: Para creacion de la interfaz y su diseño JQuery: Para manipular los elementos de la página después de que se haya cargado. — crud.js << El archivo donde se hace la lógica del CRUD y también las peticiones REST API. index.js << El archivo donde se configura el servidor NodeJS y ExpressJS index.html << El archivo HTML donde se encuentra la interfaz principal styles.css << El archivo de estilos que es aplicado en el archivo HTML

Back End

Componente	Back End
Descripción	Se encarga de administrar las peticiones de en formato REST API y devolver una respuesta al Front, manipula la base de datos directamente y retorna los cambios al Front.
Dependencias con otros componentes	Archivo de la Base de datos sql (que requiere spring configurada en una clase).
Interfaces de Salida	● **Petición GET** **Parámetros:** **body**: objeto del empleado en formato json **headers**: cabeceras de la petición: **status**: el estatus de la petición. **Descripción**: Devuelve una

	respuesta en formato json con los datos que se solicitaron a la base de datos (En este caso se solicitan todos los registros de una tabla de la base de datos). ● **Petición POST** **Parámetros:** **body**: objeto del empleado en formato json. **headers**: cabeceras de la petición. **status**: el estatus de la petición. **Descripción**: Devuelve una respuesta en formato json con los datos que se manipularon de la base de datos (En este caso se crea un nuevo registro de una tabla de la base de datos). ● **Petición PUT** **Parámetros:** **body**: objeto del empleado empleado en formato json. **headers**: cabeceras de la petición. **status**: el estatus de la petición. **Descripción**: Devuelve una respuesta en formato json con los datos que se manipularon de la base de datos (En este caso se actualiza un registro existente de una tabla de la base de datos). ● **Petición DELETE** **Parámetros:** **body**: objeto del identificador del empleado en formato json **headers**: cabeceras de la petición. **status**: el estatus de la petición. **Descripción**: Devuelve una respuesta en formato json con los datos que se manipularon de la

	base de datos (En este caso se elimina un registro existente de una tabla de la base de datos).
Interfaces de Entrada	● **Petición GET** **Parámetros:** **url:** puerto de la base de datos, nombre de la base de datos/tabla. **Tipo:** http **Descripción:** mediante el uso de la anotación RestController y GetMapping de spring, se establece el endpoint mediante el cual se podrán hacer solicitudes http (en este caso la petición GET va a obtener el contenido completo de los registros de la base de datos) en formato json y poder acceder al contenido de la base de datos a través de una respuesta que se genera en consecuencia de la solicitud. ● **Petición POST** **Parámetros:** **url:** puerto de la base de datos, nombre de la base de datos/tabla **method:** post **headers:** cabecera de la petición http con el siguiente contenido {'Accept': 'application/json', 'Content-Type': 'application/json'} **body:** objeto del empleado en formato json **Tipo:** http **Descripción:** mediante el uso de la anotación RestController y GetMapping de spring, de establecer el endpoint mediante el cual se podrán hacer solicitudes http (en este caso la petición POST crea un nuevo registro en una tabla de la base de datos) en formato json y

	poder acceder al contenido de la base de datos. ● **Petición PUT** **Parámetros:** **url:** puerto de la base de datos, nombre de la base de datos/tabla, identificador del registro. **method:** put **headers:** cabecera de la petición http con el siguiente contenido {'Accept': 'application/json', 'Content-Type': 'application/json'} **body:** objeto del empleado en formato json **Tipo:** http en formato json **Descripción:** mediante el uso de la anotación RestController y GetMapping de spring, se establece el endpoint mediante el cual se podrán hacer solicitudes http (en este caso la petición PUT actualiza un nuevo registro en una tabla de la base de datos) en formato json y poder acceder al contenido de la base de datos. ● **Petición DELETE** **Parámetros:** **url:** puerto de la base de datos, nombre de la base de datos/tabla, identificador del registro. **method:** delete **headers:** cabecera de la petición http con el siguiente contenido {'Accept': 'application/json', 'Content-Type': 'application/json'} **id:** identificador del empleado. **Tipo:** http **Descripción:** mediante el uso de la anotación RestController y GetMapping de spring, se establece el endpoint mediante el cual se podrán hacer solicitudes http (en

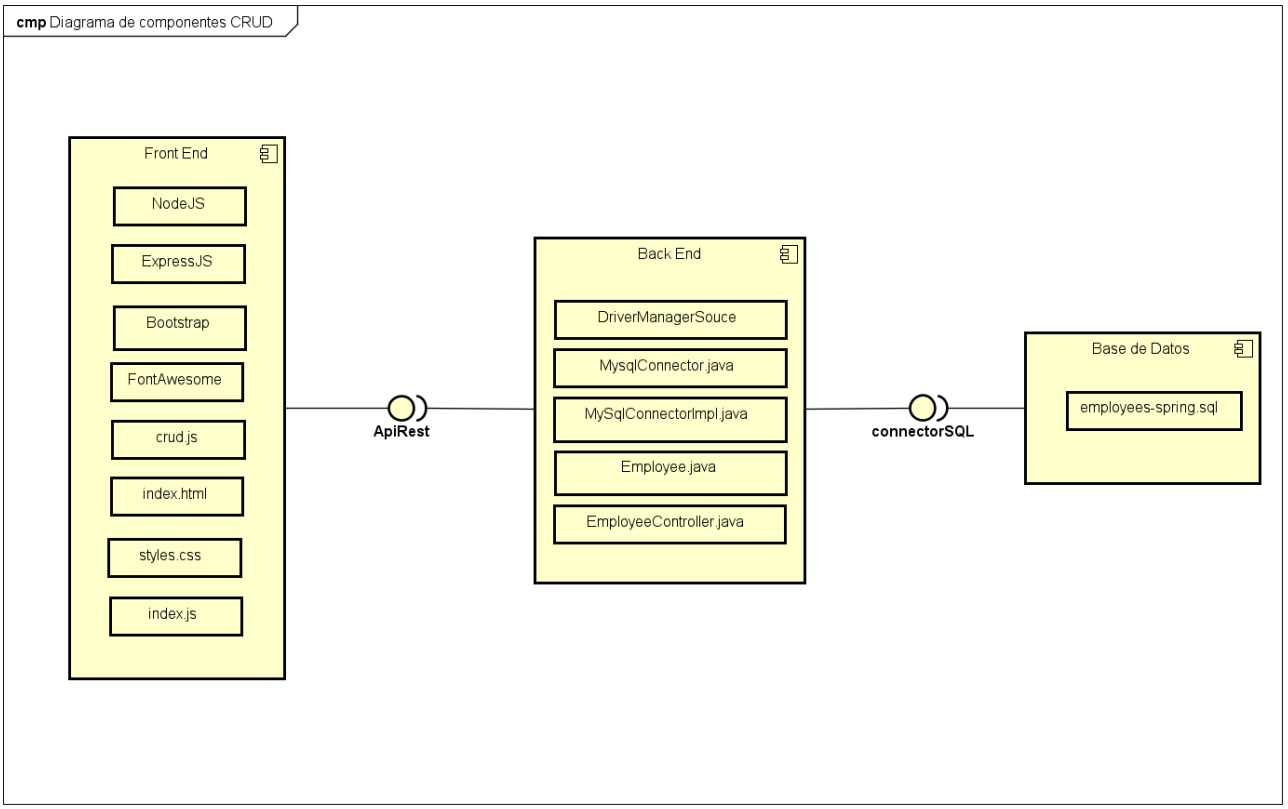
	este caso la petición DELETE eliminar registro existente en una tabla de la base de datos) en formato json y poder acceder al contenido de la base de datos.
Artefactos	**Librería:** DriverManagerSouce (Implementación de JDBC para Mysql) librería interna que provee spring para poder configurar y acceder a la base de datos. **AppConfiguration.java:** se configura el conector para que spring pueda acceder a la base de datos. **<<Interface>>MysqlConnector.java:** Interfaz que establece las operaciones que se usarán para acceder a la base de datos. **MySqlConnectorImpl.java:** implementa la librería DriverManagerSouce (para conectar a pring con la base de datos y poder manejarla) para que las funciones recuperen y manipulen la información de la base de datos. **Employee.java:** clase que abstrae a un empleado y ees usada y requerida por la clase MySqlConnectorImpl.java y **EmployeeControlle.java**r: Clase con anotación RestController que posee los métodos que reciben los endpoints y también generan la respuesta completa HTTP, mediante las cuales se manipula a través de spring la base de datos Mysql con la librería de spring DriverManagerSouce(librería JDBC para hacer consultas sql) y la clase MySqlConnectorImpl.java.

Base de Datos

Componente	Base de Datos
Descripción	La base de datos es la encargada de almacenar los registros relacionados con los empleados y suministrarlos
Dependencias con otros componentes	Ninguno
Interfaces de Salida	Información relacionada con la base de datos de empleados
Interfaces de Entrada	Información relacionada con las sentencias

	SQL de la base de datos
Artefactos	Archivo de base de datos SQL (.sql) AppConfiguration.java (clase de Spring): Se encarga de configurar spring internamente para comunicarse con la base de datos.

UML de Componentes



Descripción de Clases

Front End

Nombre de la Clase	index.js
Descripción	Configura el servidor de la página web en NodeJS con ExpressJS
Dependencias con otras clases	index.html **Tipo**: Dependencia **Descripción**: Se requiere especificar el archivo a servir al cliente por medio del servidor, en este caso, index.html. Es indirecto porque se define por medio de un String que podría cambiarse a cualquier otro archivo.
Atributos	
Número: 1 **Nombre:** express **Tipo:** module Export **Visibilidad:** public **Valor por omisión:** require("express") **Descripción:** Incluye ExpressJS en nuestro servidor en NodeJS	
Número: 2 **Nombre:** app **Tipo:** express **Visibilidad:** public **Valor por omisión:** express() **Descripción:** Instanciado de ExpressJS	
Número: 3 **Nombre:** port **Tipo:** int **Visibilidad:** public **Valor por omisión:** 8000 **Descripción:** El puerto en donde se estará sirviendo la página web.	
Funciones	
Número: 1 **Nombre:** app.use **Argumentos:** “express.static(__dirname+"/public")” (El enrutamiento de archivos estáticos a servir por express **Retorno:** ninguno **Visibilidad:** public	

Descripción: Se encarga de servir los archivos dentro de la carpeta __dirname+”/public” de forma estática al cliente.
Número: 2 **Nombre:** app.get **Argumentos:** “"/", (req, res) => {res.sendFile(__dirname+"/views/index.html");}” (El archivo a servir cuando el cliente entra a la dirección raíz del servidor) **Retorno:** ninguno **Visibilidad:** public **Descripción:** Se encarga de servir el archivo index.html cuando el cliente entra a la dirección raiz “/” del servidor.
Número: 3 **Nombre:** app.listen **Argumentos:** “port” (puerto) **Retorno:** ninguno **Visibilidad:** public **Descripción:** Se encarga de imprimir un mensaje a la consola para saber que la página web ya está escuchando en el puerto configurado.

Nombre de la Clase	crud.js
Descripción	Maneja la funcionalidad del CRUD y realiza las peticiones REST API
Dependencias con otras clases	index.html **Tipo**: Asociación **Descripción**: Sin el index.html no hay interfaz con la que enviar los eventos a las funciones dentro del crud.js. El crud.js también manipula los objetos/valores que se encuentran en index.html.
Atributos	
Número: 1 **Nombre:** seleccionados **Tipo:** Array **Visibilidad:** public **Valor por omisión:** [] **Descripción:** Arreglo en el cual se encuentran todos los números de id correspondientes a los registros seleccionados en la interfaz gráfica de la tabla.	
Funciones	
Número: 1 **Nombre:** JQ_formularioJSON	

Argumentos: “form: Form” (El formulario de la página index.html)
Retorno: “json: Array” el arreglo del json a convertir a String
Visibilidad: public
Descripción: Se encarga de generar un arreglo de los valores introducidos en el formulario para que luego este arreglo se transforme a un json con la función JSON.stringify()

Número: 2
Nombre: JQ_GET

Argumentos: ninguno
Retorno: ninguno
Visibilidad: public
Descripción: Llama a la función que hace la petición GET al REST API

Número: 3
Nombre: JQ_POST

Argumentos: “form: Form” (El formulario de la página index.html)
Retorno: “false: Boolean”
Visibilidad: public
Descripción: Llama a la función que hace la petición POST al REST API

Número: 4
Nombre: JQ_PUT

Argumentos: “form: Form” (El formulario de la página index.html)
Retorno: “false: Boolean”
Visibilidad: public
Descripción: Llama a la función que hace la petición PUT al REST API

Número: 5
Nombre: JQ_DELETE

Argumentos: “id: Integer” (El número de Id del empleado a eliminar)
Retorno: ninguno
Visibilidad: public
Descripción: Llama a la función que hace la petición DELETE al REST API

Número: 6
Nombre: cerrar_ventanaFlotante

Argumentos: ninguno
Retorno: ninguno
Visibilidad: public
Descripción: Cierra la ventana flotante del formulario

Número: 7
Nombre: mostrar_nuevoRegistro

Argumentos: ninguno
Retorno: ninguno
Visibilidad: public

Descripción: Muestra el formulario relacionado con agregar un nuevo registro y limpia los campos del mismo

Número: 8
Nombre: mostrar_modificarRegistro

Argumentos: “elemento: HTML Element” (El botón de modificar registro)
Retorno: ninguno
Visibilidad: public
Descripción: Muestra el formulario relacionado con modificar un registro existente y llena los campos del registro seleccionado a modificar para facilitar el uso

Número: 9
Nombre: tabla_agregarRegistro

Argumentos: “datos: JSON Object” (JSON Object procesado por tabla_rellenar)
Retorno: ninguno
Visibilidad: public
Descripción: Agrega un registro a la tabla a partir de un objeto JSON con las llaves y valores correspondientes

Número: 10
Nombre: tabla_rellenar

Argumentos: “getJSON : JSON” (JSON recibido por la peticion GET)
Retorno: ninguno
Visibilidad: public
Descripción: Rellena la tabla utilizando el JSON recibido por la peticion GET del REST API, agarrando las llaves y los valores del JSON e introduciendolos uno por uno a la función tabla_agregarRegistro

Número: 11
Nombre: tabla_seleccionarTodos

Argumentos: “orig: HTML Element” (el checkbox que se encuentra en la cabecera de la tabla)
Retorno: ninguno
Visibilidad: public
Descripción: Selecciona todos los checkbox y agrega sus id de registros correspondientes al arreglo “seleccionados”

Número: 12
Nombre: tabla_seleccionarElemento

Argumentos: ninguno
Retorno: ninguno
Visibilidad: public
Descripción: Selecciona el checkbox y agrega el id del registro correspondiente al arreglo “seleccionados”

Número: 13
Nombre: tabla_eliminarElemento

Argumentos: “elemento: HTML Element” (el botón de eliminar registro)

Retorno: ninguno
Visibilidad: public
Descripción: Llama a la función JQ_DELETE suministrando el id del registro obtenido.

Número: 14
Nombre: getEmpleados

Argumentos: ninguno
Retorno: ninguno
Visibilidad: public
Descripción: Hace un fetch request al URL correspondiente al GET del REST API

Número: 15
Nombre: postEmpleado

Argumentos: “jsonEmpleado: JSON Object” (el JSON Object correspondiente al nuevo registro)
Retorno: ninguno
Visibilidad: public
Descripción: Hace un fetch request al URL correspondiente al POST del REST API, y al recibir una respuesta, recarga la página

Número: 16
Nombre: putEmpleado

Argumentos:
“jsonEmpleado: JSON Object” (el JSON Object correspondiente al registro modificado)
“empleadoId: Integer” (el número de Id correspondiente al registro a modificar)
Retorno: ninguno
Visibilidad: public
Descripción: Hace un fetch request al URL correspondiente al PUT del REST API, y al recibir una respuesta, recarga la página

Número: 17
Nombre: deleteEmpleado

Argumentos: “empleadoId: Integer” (el número de Id correspondiente al registro a eliminar)
Retorno: ninguno
Visibilidad: public
Descripción: Hace un fetch request al URL correspondiente al DELETE del REST API, y al recibir una respuesta, recarga la página

Back End

Nombre de la Clase	Employee.java
Descripción	Clase que abstrae un empleado.
Dependencias con otras clases	
Ninguna	
Atributos	
Número:1 **Nombre:** id **Tipo:** Integer **Visibilidad:** private **Valor por omisión:** ninguno **Descripción:** identificador de un empleado	
Número:2 **Nombre:** nombre **Tipo:** String **Visibilidad:** private **Valor por omisión:** ninguno **Descripción:** nombre de un empleado	
Número:3 **Nombre:** email **Tipo:** String **Visibilidad:** private **Valor por omisión:** ninguno **Descripción:** correo electrónico de un empleado	
Número:4 **Nombre:** address **Tipo:** String **Visibilidad:** private **Valor por omisión:** ninguno **Descripción:** identificador de un empleado	
Funciones	
Métodos getters y setters para cada atributo	

Nombre de la Clase	AppConfiguration.java
Descripción	Clase usada para configurar la librería usada para acceder a la base de datos y configurar la forma de las llamadas.
Dependencias con otras clases	
Tipo de asociación: Generalización **Nombre de la clase:** MySqlConnectorImpl.java **Descripción:** Le provee los métodos y objetos definidos en la clase AppConfiguration.java para poder realizar las consultas sql.	
Atributos	
Ninguno	
Funciones	
Número: 1 **Nombre:** dataSource **Descripción:** Implementación simple de la interfaz JDBC DataSource estándar, configurando el viejo JDBC DriverManager a través de las propiedades del bean y devolviendo una nueva conexión de cada llamada getConnection. **Argumentos:** ninguno **Retorno:** DriverManagerDataSource **Visibilidad:** public	
Número: 2 **Nombre:** namedParameterJdbcTemplate **Descripción:** Clase de plantilla con un conjunto básico de operaciones JDBC, que permite el uso de parámetros con nombre en lugar del tradicional '?' marcadores de posición. **Argumentos:** DriverManagerDataSource: dataSource **Retorno:** NamedParameterJdbcTemplate **Visibilidad:** public	

Nombre de la Clase	EmployeeController.java
Descripción	Clase con anotación RestController que posee los métodos que reciben los endpoints y también generan la respuesta completa HTTP.

Dependencias con otras clases

Tipo de asociación: Asociación simple
Nombre de la clase: MySqlConnectorImpl
Descripción: Contiene los métodos para acceder a la base de datos dependiendo de la sentencia sql requerida.

Tipo de asociación: Asociación simple
Nombre de la clase: Employee
Descripción: Se usa para el cuerpo de la respuesta HTTP.

Atributos

Número:1
Nombre: mySqlConnectorImpl
Tipo: MySqlConnectorImpl
Visibilidad: private
Valor por omisión: ninguno
Descripción: Objeto usado para acceder a los métodos de las sentencias sql

Funciones

Número: 1
Nombre: listEmployees
Descripción: método usado para listar los empleados alojados en la base de datos a partir de una consulta http que se recibe en un endpoint y es devuelto con la lista de empleados en formato json junto a las cabeceras, cuerpo y el estatus de la petición.
Argumentos: Ninguno
Retorno: response: ResponseEntity<Employee>
Visibilidad: public

Número: 2
Nombre: createEmployee

Descripción: método usado para crear un empleado en la base de datos a partir de un objeto empleado que se recibe de un endpoint y es devuelto en formato json junto a las cabeceras, cuerpo y el estatus de la petición.
Argumentos: headers: Map<String, String> , body: String
Retorno: response: ResponseEntity<Employee>
Visibilidad: public

Número: 3
Nombre: putEmployee

Descripción: método usado para actualizar un empleado en la base de datos a partir del identificador que se recibe de un endpoint y es devuelto en formato json junto a las cabeceras, cuerpo y el estatus de la petición. **Argumentos:** id: String, **i**headers: Map<String, String> , body: String **Retorno:** response: ResponseEntity<Employee> **Visibilidad:** public
Número: 4 **Nombre:** deleteEmployee **Descripción:** método que es usado para eliminar un empleado de la base de datos a partir del identificador que se recibe de un endpoint y es devuelto en formato json junto a las cabeceras, cuerpo y el estatus de la petición. **Argumentos:**id: String, **i**headers: Map<String, String> , body: String **Retorno:** response: ResponseEntity<Employee> **Visibilidad:** public

Nombre de la Clase	<<Interface>> MysqlConnector.java
Descripción	Interfaz utilizada para crear los contratos con Mysql.
Dependencias con otras clases	
Tipo de asociación: Generalización **Nombre de la clase:** MySqlConnectorImpl.java **Descripción:** Le provee a la clase MySqlConnectorImpl.java los contratos que debe implementar para realizar las consultas de la base de datos.	
Atributos	
Ninguno	
Funciones	
Número: 1 **Nombre:** listEmployees **Descripción:** devuelve la lista de empleados alojados en la base de datos. **Argumentos:** ninguno **Retorno:** List<Employee> **Visibilidad:** public	
Número: 2 **Nombre:** createEmployee **Descripción:** crea un empleado que será alojado en un nuevo registro de la base de datos. **Argumentos:** employee: Employee **Retorno:** Map<String, Object> **Visibilidad:** public	

Número: 3 **Nombre:** updateEmployee **Descripción:** actualiza un empleado alojado en un registro de la base de datos. **Argumentos:** employee: Employee **Retorno:** Map<String, Object> **Visibilidad:** public
Número: 4 **Nombre:** deleteEmployee **Descripción:** elimina un empleado alojado en un registro de la base de datos. **Argumentos:** id: Integer **Retorno:** int **Visibilidad:** public

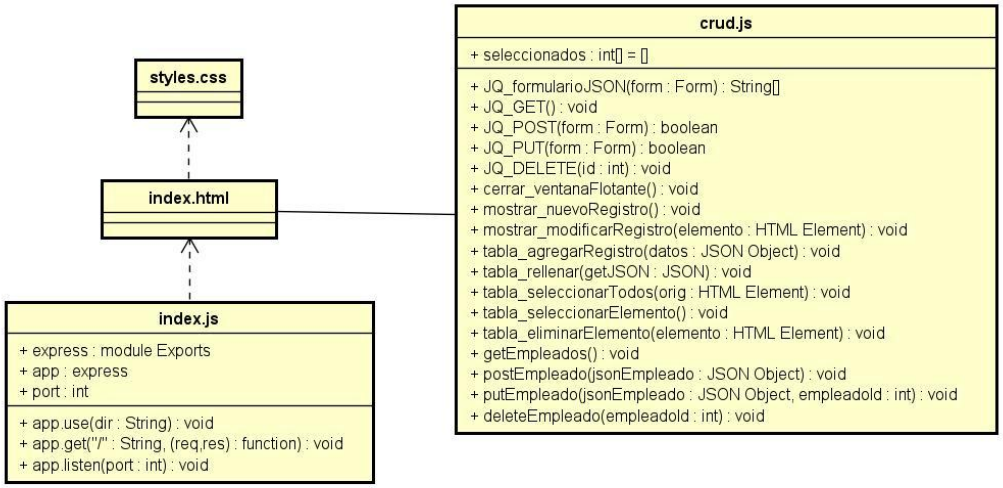
Nombre de la Clase	MysqlConnectorImpl.java
Descripción	Interfaz utilizada para crear los contratos con Mysql.
Dependencias con otras clases	
Tipo de asociación: Generalización **Nombre de la clase:** <<Interface>> MySqlConnector.java **Descripción:** Contiene los métodos para acceder a la base de datos dependiendo de la sentencia sql requerida.	
Tipo de asociación: Asociación Simple **Nombre de la clase:** Employee.java **Descripción:** Utiliza objetos de la clase Employee.java para poder realizar las consultas sql.	
Tipo de asociación: Composición **Nombre de la clase:** AppConfiguration.java **Descripción:** Utiliza los objetos(DriverManagerDataSource y NamedParameterJdbcTemplate) definidos en las funciones de la clase AppConfiguration.java para poder realizar las consultas sql.	
Atributos	
Número:1 **Nombre:** namedParameterJdbcTemplate **Tipo:** NamedParameterJdbcTemplate **Visibilidad:** private **Valor por omisión:** ninguno **Descripción:** Variable usada para configurar las sentencias SQL mediante la plantilla con un conjunto básico de operaciones JDBC.	
Número: 2 **Nombre:** GET_LAST	

Tipo: String **Visibilidad:** private **Valor por omisión:** Sentencia sql **Descripción:** Sentencia sql para obtener el último registro agregado de un empleado.
Número: 2 **Nombre:** GET **Tipo:** String **Visibilidad:** private **Valor por omisión:** Sentencia sql **Descripción:** Sentencia sql para obtener todos los registros de la tabla de empleados de la base de datos.
Número: 1 **Nombre:** POST **Tipo:** String **Visibilidad:** private **Valor por omisión:** Sentencia sql **Descripción:** Sentencia sql para crear un registro de un empleado.
Número:1 **Nombre:** PUT **Tipo:** String **Visibilidad:** private **Valor por omisión:** ninguno **Descripción:** Sentencia sql para actualizar un registro de un empleado.
Número:1 **Nombre:** DELETE **Tipo:** String **Visibilidad:** private **Valor por omisión:** ninguno **Descripción:** Sentencia sql para eliminar un registro de un empleado.
Funciones
Número: 1 **Nombre:** listEmployees **Descripción:** devuelve la lista de empleados alojados en la base de datos. **Argumentos:** ninguno **Retorno:** List<Employee> **Visibilidad:** public
Número: 2 **Nombre:** createEmployee **Descripción:** Devuelve una lista de empleados a partir del mapa que se recibe como entrada. **Argumentos:** map: Map<String, Object> **Retorno:** Employee

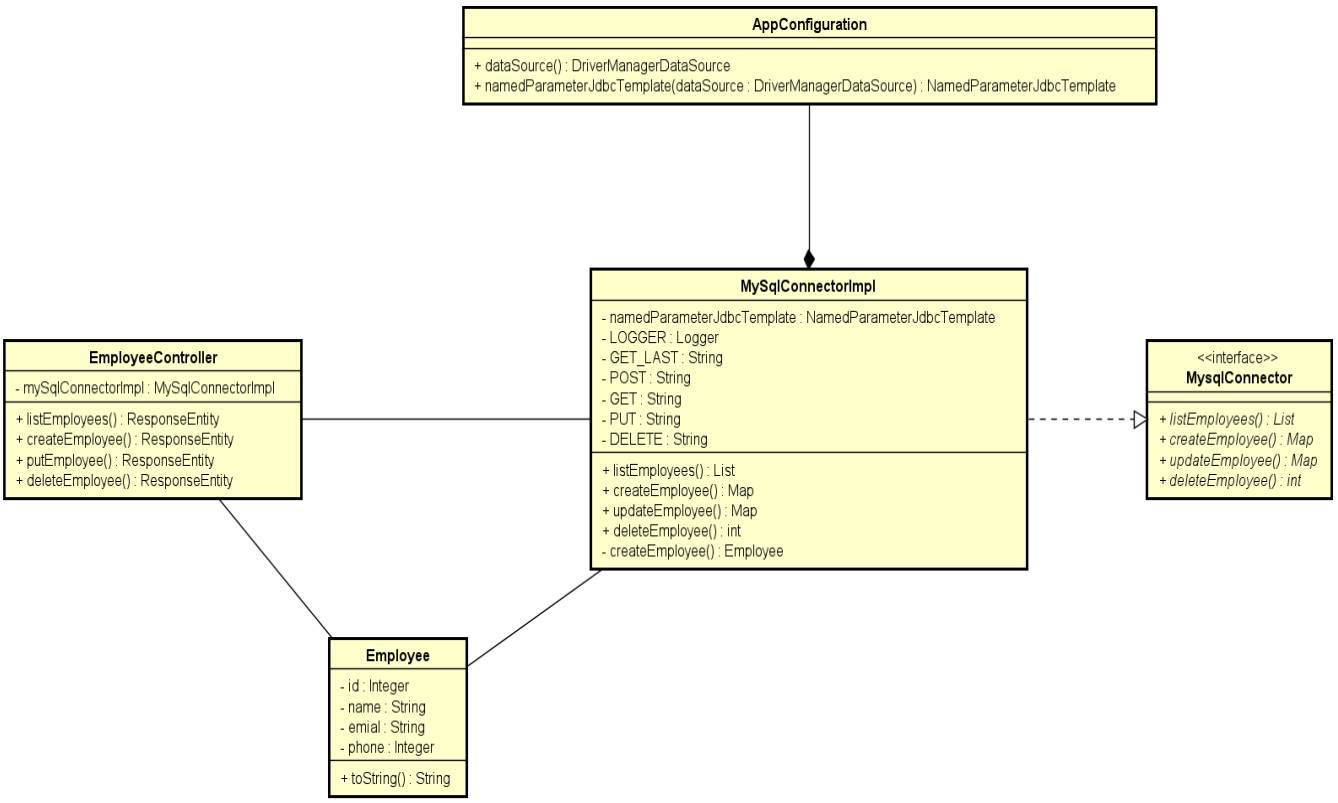
Visibilidad: private
Número: 3 **Nombre:** createEmployee **Descripción:** crea un empleado que será alojado en un nuevo registro de la base de datos. **Argumentos:** employee: Employee **Retorno:** Map<String, Object> **Visibilidad:** public
Número: 4 **Nombre:** updateEmployee **Descripción:** actualiza un empleado alojado en un registro de la base de datos. **Argumentos:** employee: Employee **Retorno:** Map<String, Object> **Visibilidad:** public
Número: 5 **Nombre:** deleteEmployee **Descripción:** elimina un empleado alojado en un registro de la base de datos. **Argumentos:** id: Integer **Retorno:** int **Visibilidad:** public

UML de Clases

Front End



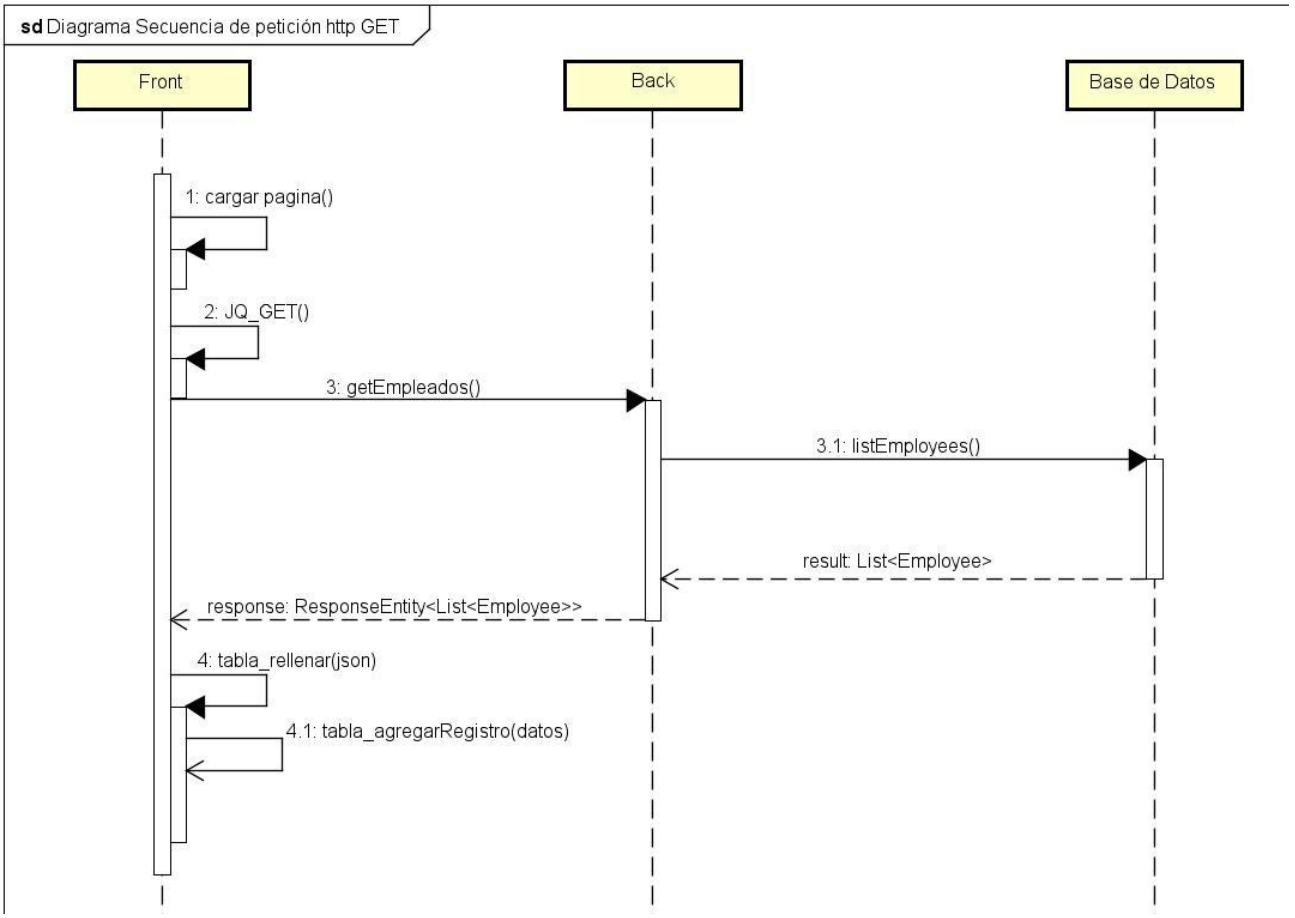
Back End


AppConfiguration
+ dataSource() : DriverManagerDataSource
+ namedParameterJdbcTemplate(dataSource : DriverManagerDataSource) : NamedParameterJdbcTemplate
MySqlConnectorImpl
- namedParameterJdbcTemplate : NamedParameterJdbcTemplate
- LOGGER : Logger
- GET_LAST : String
- POST : String
- GET : String
- PUT : String
- DELETE : String
+ listEmployees() : List
+ createEmployee() : Map
+ updateEmployee() : Map
+ deleteEmployee() : int
- createEmployee() : Employee
EmployeeController
- mySqlConnectorImpl : MySqlConnectorImpl
+ listEmployees() : ResponseEntity
+ createEmployee() : ResponseEntity
+ putEmployee() : ResponseEntity
+ deleteEmployee() : ResponseEntity
<<interface>>
MysqlConnector
+ listEmployees() : List
+ createEmployee() : Map
+ updateEmployee() : Map
+ deleteEmployee() : int
Employee
- id : Integer
- name : String
- emial : String
- phone : Integer
+ toString() : String

Descripción de las secuencias

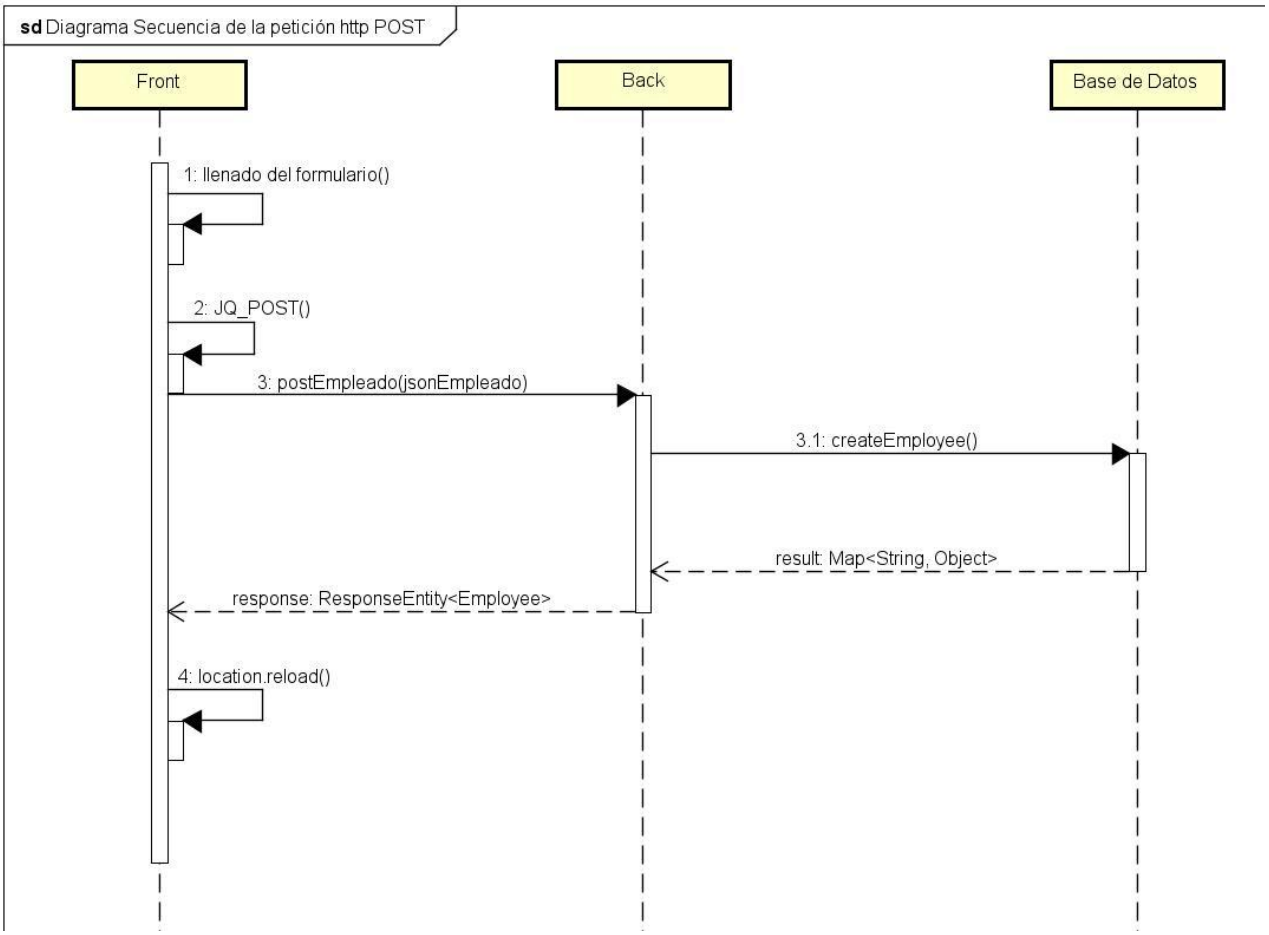
Obtener la lista de empleados y mostrarla al usuario

Descripción: Al cargar la página, se realiza la petición GET al REST API para obtener del back end los registros de los empleados que se encuentran en la base de datos.



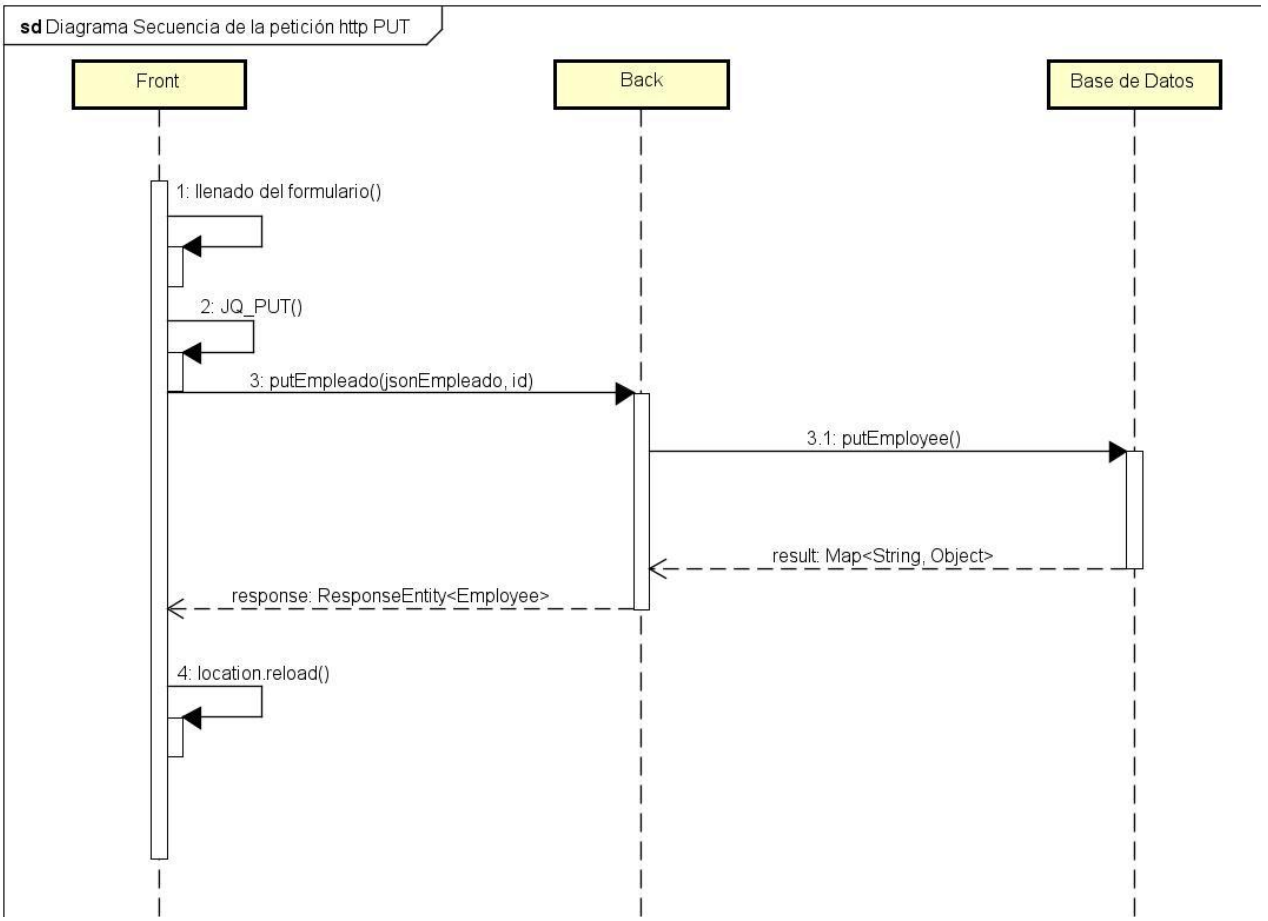
Agregar un registro a la lista de empleados y mostrarlo al usuario

Descripción: Al llenar el formulario, se realiza la petición POST al REST API para enviarle los datos al back end y que este agregue el nuevo registro a la base de datos. Después de realizar eso, recarga la página.



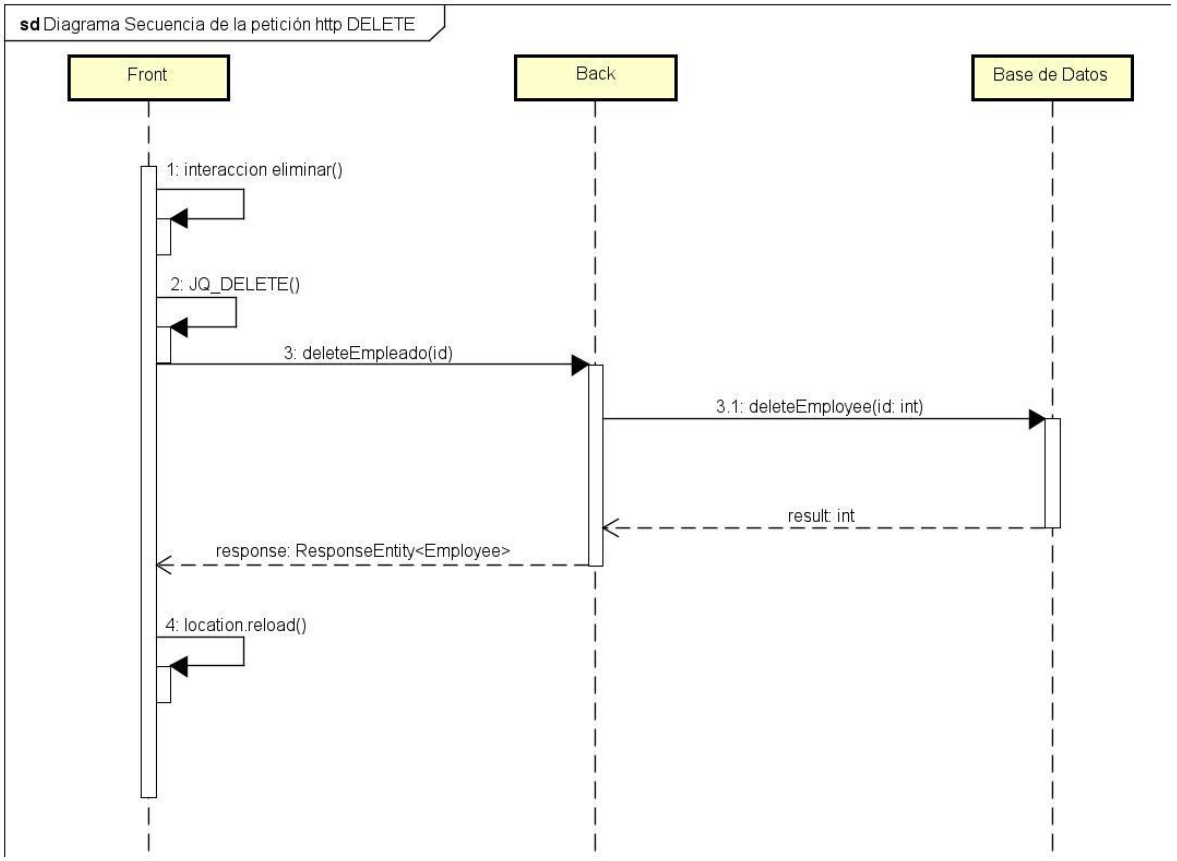
Modificar un registro de la lista de empleados y mostrar los cambios al usuario

Descripción: Al llenar el formulario, se realiza la petición PUT al REST API para enviarle los datos al back end y que este modifique el registro deseado en la base de datos. Después de realizar eso, recarga la página.



Eliminar un registro de la lista de empleados y mostrar los cambios al usuario

Descripción: Al hacer click a eliminar (individualmente) un registro, se realiza la petición DELETE al REST API para enviarle el número de id del empleado a eliminar al back end y que este elimine el registro de la base de datos. Después de realizar eso, recarga la página.



Descripción de caso de uso

1.- Agrega un nuevo registro
2.- Modifica un registro existente
3.- Elimina un registro individual
4.- Seleccionar el resto de los registros
5.- Elimina los registros seleccionados con el botón de eliminar seleccionados

Enlace Youtube

https://youtu.be/zJAIfLfg8Ys

Otras funcionalidades presentes

1.- Excepciones DuplicateKeyException y DataAccessException en Spring
2.- Ventana flotante del formulario en el Front End