PROYECTO INTEGRADOR

DOCUMENTACIÓN CRUD EN EXPRESS

ARQUITECTURA DE SOFTWARE.

QUINTO SEMESTRE.

DOCENTE: VÍCTOR HUGO MENÉNDEZ DOMÍNGUEZ.

INTEGRANTES:

AKÉ GAMBOA ALEJANDRO

FERNÁNDEZ MENA ARIEL DE JESÚS

GAMBOA CAMPOS WILBERTH MANUEL

IZUNZA NAJAR LUIS ANTONIO

PACHECO SERRALTA ESTEBAN ALFONSO.

FECHA: 9 DE DICIEMBRE DE 2022

DESCRIPCIÓN DE FUNCIONALIDADES.

Nombre	Objetivo principal	Funcionalidades
Tailwind.config.js	Importar los modelos correspondientes; los cuales son: formUser*.js Index*.js	Exportar módulos.

Nombre	Objetivo principal	Funcionalidades
Postscss.config.js	Importar de componentes css: talwindcss stylecss	Exportar módulos.

Nombre	Objetivo principal	Funcionalidades
Index.js	Poner en funcionamiento a componente server.	Llamar a server.listen()

Nombre	Objetivo principal	Funcionalidades
Config.js	Establecer una conexión con la base de datos local	Leer puerto, host, usuario, contraseña y nombre de la base de datos.

Nombre	Objetivo principal	Funcionalidades
View.route.js	Mandar petición a controlador view.controller.js	Comunicación con el controlador view.controller.js para mandar la petición agregar.

Nombre	Objetivo principal	Funcionalidades

User.js	Validar campos que han sido ingresados desde la vista agregar.html.	-validar campo nombre. -validar campo número. -validar campo número de celular. -validar ID.

Nombre	Objetivo principal	Funcionalidades
domIndex.js	Manejar el DOM de la vista index.	-Eliminar usuario de la tabla dentro de la vista index. -Agregar usuario de la tabla dentro de la vista index. -pasar a siguiente página de la tabla. -pasar a la anterior página de la tabla. -buscar dentro de la tabla si se encuentra lo escrito dentro del navegador.

Nombre	Objetivo principal	Funcionalidades
appIndex.js	Iniciar ejecuciones necesarias para el funcionamiento dinámico de la vista index.	-iniciar consultas en tabla. -iniciar listado. -iniciar eliminar un usuario. -iniciar en pasar a siguiente página. -iniciar pasar a anterior página. -iniciar buscar usuario en tabla.

Nombre	Objetivo principal	Funcionalidades
requestFormUser.js	Manejar los estados dentro del servidor de acuerdo a lo que fue solicitado en DOM.	-manejar solicitudes.

Nombre	Objetivo principal	Funcionalidades
domFormUser.js	Generar alertas de acuerdo a los resultados correspondientes de las acciones hechas.	-alertar por agregar usuario. -alertar errores. -obtener valores. -agregar usuario.

Nombre	Objetivo principal	Funcionalidades
appFormUser.js	Agregar evento a la tabla.	-Generar evento luego de realizar un click.

Nombre	Objetivo principal	Funcionalidades
Api.js	Encargada de llamar a la ApiRest	-consultar Api. -postApi. -realizar una petición post.

Nombre	Objetivo principal	Funcionalidades
User.model.js	Almacenar de forma temporal los datos de usuario.	-Guardar nombre. -Guardar número.

Nombre	Objetivo principal	Funcionalidades

Server.js	Habilitar al servidor.	-conectar a base de datos. -middleware. -respuesta apertura. -reclamar elementos a viajar.

Nombre	Objetivo principal	Funcionalidades
reqValidator.middleware.js	Validación del request en el routing antes de entrar a los controladores.	-validar campos.

Nombre	Objetivo principal	Funcionalidades
Config.js	Determinar el dialecto de la base de datos.	-determinar dialecto del que se basa la base de datos.

Nombre	Objetivo principal	Funcionalidades
View.Controller.js	Recepcionar los datos recibidos de las vistas.	-importar valores de las vistas. -enviar datos a los que requieran usarlo.

Nombre	Objetivo principal	Funcionalidades
Users.controller.js	Encapsulamiento de las variables que entraron.	-enviar usuarios. -cancelar usuarios.

DESCRIPCIÓN DE COMPONENTES.

Nombre: Tailwind.config.js

Descripción: Importar los modelos correspondientes a los paquetes los cuales son:

- formUser*.js
- Index*.js

Dependencias con otros componentes:

Index.js:

- eliminarUsuarioDom()
- listarUsuarioDom()
- eventoBtnSiguiente()
- eventoBtnAnterior()
- eventoBtnBuscar()
- appIndex()

formUser.js:

- alertAgregarUsuario()
- alertErrores()
- obtenerValoresForm()
- agregarUsuario()
- manejadorSolicitudes()

Interfaces de Salida: tailwind.config(realiza un llamado a los servicios de Index.js y formUser.js para su salida)

Interfaces de Entrada:

Evento(lo que se envia es el tipo de evento que ha sido invocado si en su caso es alertar agregación usuario o eliminación de usuario, errores, paginación siguiente o anterior y buscar)

Artefactos:

- Agregar.html
- Index.html

Nombre: Postscss.config.js

Descripción: es la configuración de elementos css.

Dependencias con otros componentes: N/A

Interfaces de Salida: configuración (luego de la recepción, envía los resultados)

Interfaces de Entrada: Estilos CSS (se envían enlaces desde las hojas css de las que posee una relación)

Artefactos:

- tailwindcss.css
- style.css

Nombre: index.js

Descripción: prepara el llamado al servidor.

Dependencias con otros componentes:

- Listen() (agarrado de server.js)

Interfaces de Salida: N/A

Interfaces de Entrada:

- Listen() (recibe el estado en el que se encuentra el servidor, lo esperado es que esté abierto)

Artefactos: N/A

Nombre: config.js

Descripción: describe los parámetros de entrada que son correspondientes a la base de datos.

Dependencias con otros componentes: N/A

Interfaces de Salida:

- dirección base de datos

Interfaces de Entrada: N/A

Artefactos:

- simple.mysql

Nombre: view.route.js

Descripción: enmarca la ruta para acceder al controlador view.controller.js

Dependencias con otros componentes:

view.controller.js

- Index()

- Agregar()

Interfaces de Salida:

- Route() (la ruta de acceso)

Interfaces de Entrada:

- Index()
- Agregar()

Artefactos: N/A

Nombre: user.js

Descripción: validar los campos que han sido registrados desde la ventana vista.html

Dependencias con otros componentes:

Users.controller.js:

- userGet()
- userPost()
- userDelete()

ReqValidator.middleware.js:

- validarCampos()

Interfaces de Salida:

- aceptación(si los datos no tuvieron problemas, se van, caso contrario, no se aceptan y no se exportan)

Interfaces de Entrada:

- userGet() (recepción de los valores que entran)
- userPost() (se rechazan en caso que alguno esté vacío)
- userDelete() (valida si los datos son correctos)

Artefactos:

- express, express-validator

Nombre: DomIndex.js

Descripción: se encarga de manejar el DOM de la vista index.html

Dependencias con otros componentes:

Api.js:

- consultarApi()

Interfaces de Salida: evento(sale el resultado logrado ya sea agregar, eliminar, búsqueda realizado)

Interfaces de Entrada:

- consultarApi()

Artefactos: N/A

Nombre: appIndex.js

Descripción: Iniciar ejecuciones necesarias para el funcionamiento dinámico de la vista index.

Dependencias con otros componentes:

Api.js:

- consultarApi()

DomIndex.js:

- listarUsuariosDom()
- eliminarUsuarioDom()
- eventoBtnAnterior()

- eventoBtnSiguiente()
- eventoBtnBuscar()

Interfaces de Salida: N/A

Interfaces de Entrada:

- Evento (el componente recibe un evento de DoIndex.js y a partir de eso ejecuta el evento con información)

Artefactos: N/A

Nombre: requestFormUser.js

Descripción: dependiendo de la acción solicitada, llama al evento mandado del DOM

Dependencias con otros componentes:

domFormUser.js:

- alertAgregarUsuario()
- alertarErrores()

Interfaces de Salida:

- manejadorSolicitudes() (se envía el estado en el que se encuentra la información si está correcta o si tiene errores)

Interfaces de Entrada:

- alertAgregarUsuario() (en caso que la información del DOM esté correcta, se da paso en enviar una alerta de que ha sido agregado)
- alertarErrores() (si lo que se analizó no es correcto, se alerta que no está bien)

Artefactos:agregar.html e index.html

Nombre: domFormUser.js

Descripción: generar alertas y manipular el DOM de la vista agregar.

Dependencias con otros componentes:

Api.js:

- postApi()

requestFormUser.js

- manejaroSolicitudes()

Interfaces de Salida:

- alertGuardarUsuario() (en caso que la información del DOM esté correcta, se da paso en enviar una alerta de que ha sido agregado)
- alertErrores() (si lo que se analizó no es correcto, se alerta que no está bien)

Interfaces de Entrada:

- postApi() (la promesa de almacenar los datos es enviada para determinar su próximo estado)

Artefactos: N/A

Nombre: appFormUser.js

Descripción: generar alertas y manipular el DOM de la vista agregar.

Dependencias con otros componentes:

domFormUser.js:

- obtenerValores()

Interfaces de Salida:

- evento(se guarda el evento)

Interfaces de Entrada:

- obtenerValores() (la promesa de almacenar los datos es enviada para determinar su próximo estado)

Artefactos: N/A

Nombre:api.js

Descripción: se encarga de hacer el llamado al apiRest

Dependencias con otros componentes: N/A

Interfaces de Salida:

- consultarApi()(realizar una petición get para obtener a los usuarios)
- postApi()(la promesa de almacenar los datos es enviada para determinar su próximo estado)
- deleteApi(realizar una petición al post)

Interfaces de Entrada: N/A

Artefactos: N/A

Nombre: user.model.js

Descripción: desde la base de datos, agarra los valores de los usuarios

Dependencias con otros componentes: config.js

Interfaces de Salida:

- User(valores de salida que salen directo de la base de datos como lo es el id y el número)

Interfaces de Entrada:

- Dirección de base de datos

Artefactos: N/A

Nombre: server.js

Descripción: clase que alza el servidor local.

Dependencias con otros componentes:

- view.route.js:
  route()

Interfaces de Salida:

- Listen() (activar el puerto)

Interfaces de Entrada:

  Route() (se recibe la petición para agregar usuarios)

Artefactos: express

Nombre: reqValidator.middleware.js

Descripción: Realiza la validación de la request en el routing antes de entrar al controller de lo contrario dentro del controller se tendría que verificar que no existieran errores, haciendo el código repetitivo.

Dependencias con otros componentes:

FormUser.js:

- alertGuardarUsuario() (en caso que la información del DOM esté correcta, se da paso en enviar una alerta de que ha sido agregado)
- alertErrores() (si lo que se analizó no es correcto, se alerta que no está bien)

Interfaces de Salida:

- validarCampos(se ingresan parámetros y en base a lo que arrojen, se envía un mensaje de aceptación o rechazo)

Interfaces de Entrada:

- alertGuardarUsuario() (en caso que la información del DOM esté correcta, se da paso en enviar una alerta de que ha sido agregado)
- alertErrores() (si lo que se analizó no es correcto, se alerta que no está bien)

Artefactos: express-validator

Nombre:view.controller.js

Descripción: recepcionar las entradas de las vistas html para su uso.

Dependencias con otros componentes: N/A

Interfaces de Salida:

- Index(envía el número del índice a quien lo requiera)
- Agregar(provee los datos ingresados a quien lo necesite)

Interfaces de Entrada:

- Index
- agregar

Artefactos:

- Index.html
- Agregar.html

Nombre:user.controller.js

Descripción: encapsulamiento de las variables que entran al sistema, los cuales son nombre y número.

Dependencias con otros componentes:

User.model.js:

- User

Interfaces de Salida:

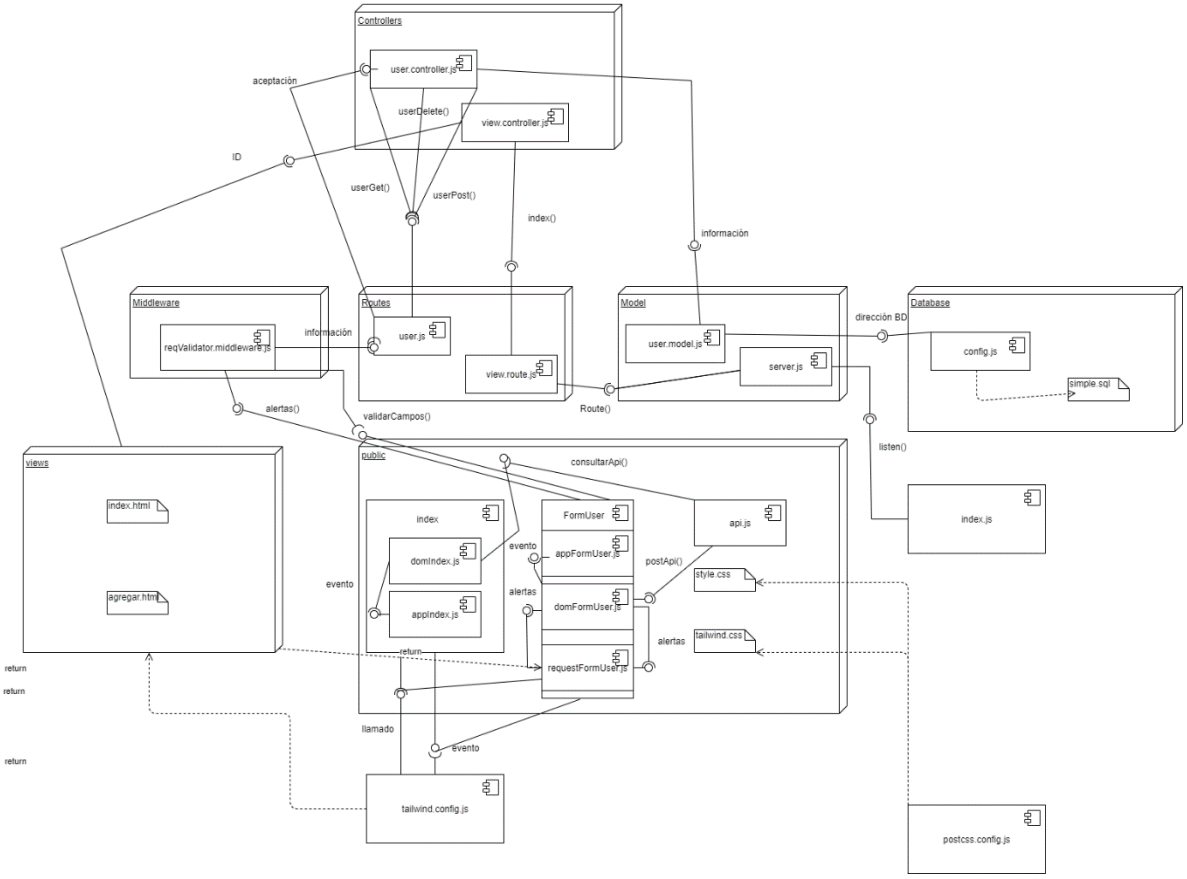
- userGet()(aceptar la petición)
- userPost()(cancelar la petición)
- userDelete()(eliminar la petición)

Interfaces de Entrada:

- User(valores de salida que salen directo de la base de datos como lo es el id y el número)

Artefactos:express-validator.

DIAGRAMA DE COMPONETES:



DOCUMENTACIÓN DE CLASES.

Nombre de la clase: index.js

Descripción: poner en funcionamiento el servidor de la aplicación.

Dependencias con otras clases: server.js (se usa un llamado para activar el servidor)

Atributos: server(object,public) un objeto server invocado para activar el servidor.

Funciones: N/A

Nombre de la clase: config.js

Descripción: datos pertenecientes a la base de datos.

Dependencias con otras clases: N/A

Atributos:

- PORT(String, public,3000) el dirección Puerto localhost
- DB_HOST(String, public, localhost)el tipo de host, que en este caso es local.
- DB_USER(String, public,root) nombre del usuario de la base de datos.
- DB_PASSWORD(String, public, null) la contraseña para acceder a la base de datos.
- DB_NAME(String, public,simple) nombre de la base de datos.

Funciones: N/A

Nombre de la clase: view.route

Descripción: enmarca la ruta para acceder al controlador view.controller.js

Dependencias con otras clases: view.controller.js

Atributos: router(Object,public) permitir hacer viajar la petición de agregar.

Funciones: N/A

Nombre de la clase: Api

Descripción: Clase encargada de llamar a la ApiRest.

Dependencias con otras clases: Ninguna

Atributos: Ninguno

Funciones:

I. (async) deleteApi(id) → {Promise}
II. (async) getApi() → {Promise}
III. (async) postApi(userRequest) → {Promise}
IV. (async) putApi(id, userRequest) → {Promise}

Nombre de la clase: AlertasSweet.

Descripción: Clase encargada de manejar las alertas proporcionadas por SweetAlert2.

Dependencias con otras clases: Api

Atributos: Ninguno

Funciones:

I. alertarNoResultados (consulta) → {void}
II. alertarSolicitudNoProcesada (consulta)
III. (async) alertarUsuarioEliminado () → {Promise}

Nombre de la clase: DomFormUser

Descripción: Clase encargada de generar alertas y manipular el dom en la vista Agregar.

Dependencias con otras clases: alertasSweet

Atributos: Ninguna

Funciones:

I. Clase encargada de generar alertas y manipular el dom en la vista Agregar.
II. alertAgregarUsuario () → {void}
III. alertarErrores (resJson) → {void}
IV. obtenerValoresForm () → { UserRequest }

Nombre de la clase: índicedom

Descripción: Clase encargada de manejar el dom de la vista index.

Dependencias con otras clases: SweetAlert

Atributos: Ninguna

Funciones:

I. eliminarConCheckBox () → {void}
II. eventoBtnAnterior () → {void}
III. eventoBtnBuscar () → {void}
IV. eventoBtnSiguiente () → {void}
V. inicializarEliminacionyEdicionUsuario () → {void}
VI. listarUsuarios (res) → {void}
VII. marcarCheckBox () → {void}
VIII. procesoActualizarDom () → {void}
IX. verificarCheckBoxNavegacion () → {void}

Nombre de la clase: RequestFormUser.

Descripción: Clase encargada de manejar los estatus.

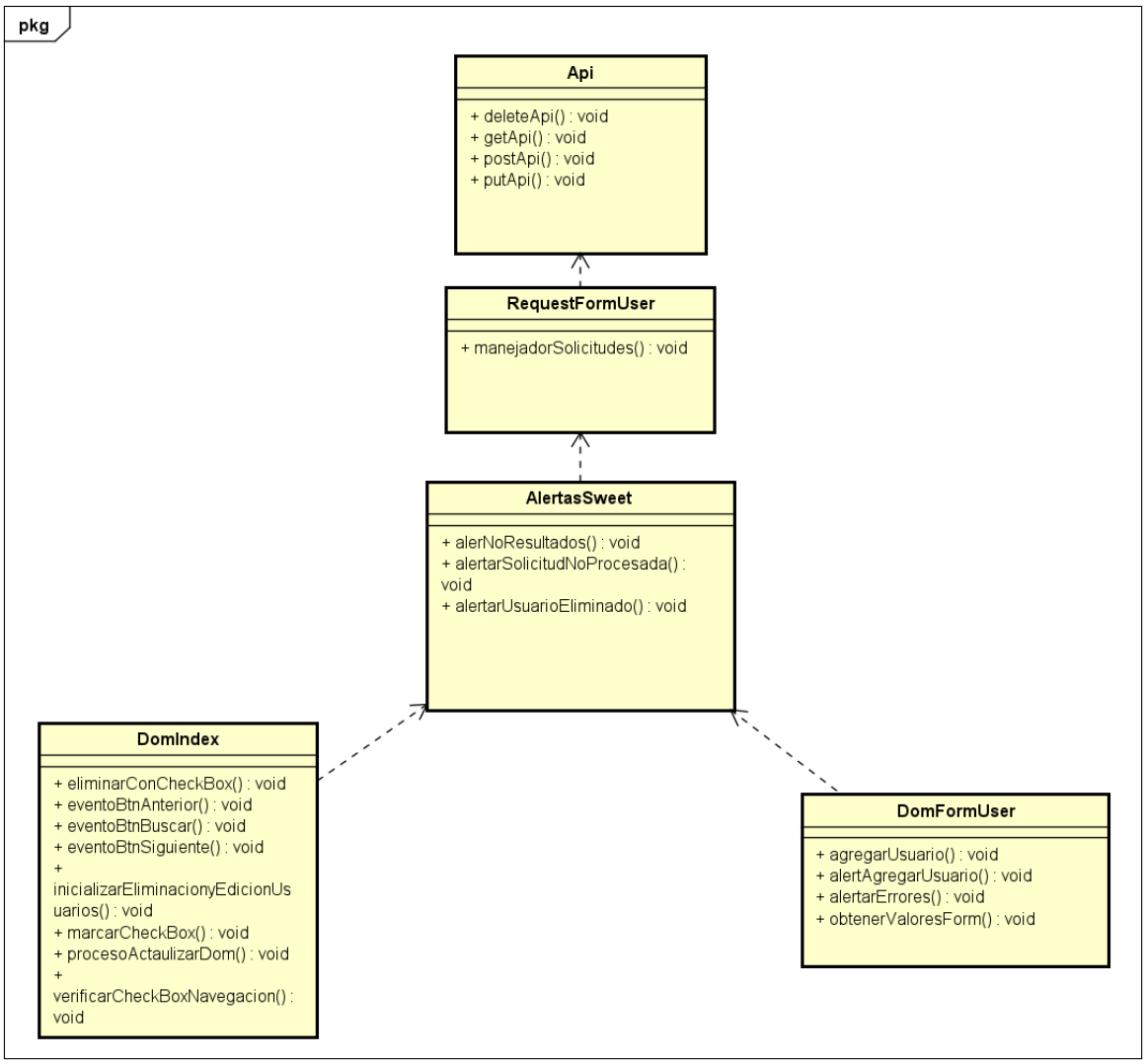
Dependencias con otras clases: Api

Atributos: Ninguno

Funciones

I. (async) manejadorSolicitudes (promise) → {void}

Añadir diagrama de clases



DESCRIPCIÓN DE LAS SECUENCIAS.

Caso de uso eventos.

Descripción: En este caso de uso se muestra la secuencia de los eventos que son seleccionados, validados y retroalimentados. Los eventos son los de agregar, eliminar, buscar y paginación.