Universidad Nacional Autónoma de México.

Facultad de Ciencias.

Modelado y programación grupo 7054

Proyecto 1

Equipo “el sazón de la programación”.

Integrantes:

-Medina Peralta Joaquín.

-Eslava Mendoza Fernando.

-Comas Castañeda Mauricio Santiago.

Proyecto 1: Consulta de clima de aeropuerto

Los Web Services permiten a las organizaciones intercambiar datos sin necesidad de conocer los detalles de sus respectivos Sistemas de Información. En este proyecto se deberá hacer uso de estas herramientas para llevar a cabo una aplicación que consulte en tiempo real el clima de ciudades dadas.

Objetivo:

Este proyecto se basa en la utilización de Web Services para desarrollar una aplicación innovadora que permite a los usuarios acceder en tiempo real a información climática de ciudades específicas. El objetivo principal es proporcionar datos precisos y actualizados sobre el clima de origen y destino para vuelos programados en el mismo día. Esta aplicación está diseñada con una interfaz intuitiva y accesible, dirigida tanto a usuarios técnicos como a aquellos que no tienen experiencia en programación.

Características Principales de la Aplicación

Entradas Versátiles: La aplicación acepta una variedad de tipos de entrada, incluyendo el número de ticket y las coordenadas de latitud y longitud de origen y destino. Además, implementa un enfoque tolerante a errores, lo que significa que los usuarios pueden ingresar nombres de ciudades mal escritos sin problemas.

Obtención de Datos Climáticos: Aprovechando los servicios web, la aplicación consulta la API de OpenWeatherMap para recopilar información climática precisa en tiempo real. Esto garantiza que los usuarios tengan acceso a datos actualizados y relevantes.

Experiencia de Usuario Amigable: La interfaz de usuario está diseñada para ser fácil de usar y comprender. Esto es especialmente importante dado que el público objetivo incluye a sobrecargos, pilotos y clientes promedio que no necesariamente tienen experiencia en programación.

Entorno de trabajo:

En este proyecto, se trabajara con MERN stack, el cual esta pensado para aplicaciones web con las siguientes herramientas:

- MongoDataBase: Es una aplicacion de base de datos no relacional, que guarda la informacion en archivos JSON y contiene su propia sintaxis para realizar las consultas. Su filosofia consiste en la informacion guardadas en documentos para su facil consulta.
- Express: Es un frameworks para aplicaciones web para node js, el cual se encarga de construir una sólida infrastructura y con el se puede el manejo de rutas, webpack, errores y backend para el llamado a la aplicacion de node js.
- React: Es una libreria mantenida y desarrollada por Facebook para el desarrollo de frontend para darle más limpieza al código, organizacion y mejor interaccion entre el usuario y el servidor.

- Node Js: Es un entorno de tiempo de ejecucion para la parte del servidor, basando su sintaxis en Javascript y en el control de eventos, y diseñado para la simplificacion de la comunicacion debido a que no tiene subprocesos y de esta manera aprovechar multiples nucleos en su entorno y compartir sockets entre procesos.
- Dotenv: Una herramienta de entorno que sirve para ocultar claves de acceso que no querriamos que fueran visibles, vease claves de api la uri de consulta de mongo, los identificadores de las colecciones, nombre de la base de datos etc.
- Jasmine: Una herramienta para los unit test en javascript que proporciona atajos para que el diseño de las pruebas sea mas facil.
- Levenshtein: Un algoritmo que detecta y corrige errores tipográficos en las ciudades ingresadas por los usuarios y encuentra coincidencias cercanas cuando los nombres de las ciudades no son exactos. Esto garantiza que incluso con entradas de usuario con errores, la aplicación pueda proporcionar datos climáticos precisos.
- Node-cron: Es una biblioteca utilizada para programar tareas periódicas (cron jobs) en la aplicación, como el registro de consultas y la actualización de datos climáticos.
- OpenWeather : Una API de obtencion de informacion del clima de una ciudad.

Entendiendo el problema:

-¿Que es lo que queremos obtener? El clima de alguna ciudad dado un nombre de ciudad o un ticket o un un IATA.

-¿Cuales son los datos que tenemos para obtenerlo? tenemos algunas bases de datos que tendran tickets asociados a ciudades en codigo IATA, y ciudades en codigo IATA.

-¿Son suficientes? Sirven para poder trabajar y pensar en una solucion para el problema, asi que es un si.

-¿Que hace que el resulado obtenido resuelva el problema? Que el programa sea capaz de mostrar dadas nuestras entradas un clima que el ususario indique y a su vez que el usuario pueda operar el programa sin problema.

Requisitos funcionales y no funcionales:

Funcionales:

- Datos de entrada:
    - Un cadena de texto de 16 caracteres que representa el identificador de un ticket de la parte visual del programa por parte del usuario y del servidor un archivo csv
    - Una cadena de texto con el nombre de la ciudad que desea obtener su clima de la parte visual del programa por parte del usuario y del servidor un archivo csv
- Datos de salida:

- El clima de las ciudades consultadas del usuario, en un json con destino al visualizador del programa

No funcionales:

- Eficiencia: se busca la obtencion de los datos de manera rápida, en especifico en complejidad O(n)
- Tolerancia a fallas. se busca que el programa sea robusto ante possibles errores por parte del usuario, es decir al ingresar el nombre de una ciudad, se espera que aunque este mal, se encuentre aun asi al clima más parecido
- Amigabilidad: se busca que el usuario pueda interactuar de manera sencilla al programa.
- Escabilidad: solo es necesario una aplicacion que pueda regresar el clima por el momento.
- Seguridad : es necesario que la informacion de los tickets no sea visible ante el usuario debido a la importancia de estos, ya que se puede encontrar toda la informacion de cada vuelo.

Para ejecutar el programa

Las indicaciones para su compilacion como sus especificaciones se encuentran en el archivo README.txt

Posibles Errores

- Si encuentras problemas al ejecutar la aplicación y recibes un error relacionado con la falta del módulo `dotenv`, asegúrate de haber creado correctamente el archivo . `env` con las variables de entorno necesarias (para mayor detalles en el README.txt).
- Un posible error es a la conexion a la base de datos de mongodb, para la compilacion del código se utilizo una base de datos de ATLAS para su ejecucion, y de este modo requirio conexión a internet.

Mantenimiento a futuro de la aplicacion

Debido a que se diseño la aplicacion con MERN stack, contamos con la ventaja de que es un lenjuage de programacion que se mantiene actualizado y con un soporte de una gran comunidad como tambien es el caso de REACT que es sustentado por Meta, de este modo no nos preocupariamos de problemas de mantenimiento a futuro

Los puntos a tocar sobre el mantenimiento futuro y propuestas de la aplicacion son las siguientes:

- Implementar el framework de REACT para que pueda ser llevado de manera nativa con REACT native y de este manera a aplicaciones de telefonos y en un posible futuro proyecto, ser agregado a una aplicacion web o aplicacion de télefono con la información importante del aereopuerto de la ciudad de méxico y este al dia con los cambios de tecnología.
- Una interfaz para el administrador de la base de datos de mongodb para ver posibles errores, datos del clima de los lugares dados de alta, actualizacion de la informacion de la ciudad,

visualizarinformacion y actualizacion de los tickets, debido a que es informacion que nunca se mantiene igual entonces es necesario verificar su veracidad.

Proceso de solucion de problemas

A continuacion se presenta el pseudocodigo de la aplicacion enfocandose en las principales ramas del desarrollo web, los cuales son: vista, modelo y controlador

Pseudo codigo para los metodos de vista (clase visualizador):

Funcion cargarClima, recibe una referencia String con nombre url.

```
variable response <- metodo fetch(url)
  Si response.status != 200{
  throw new Error("Error al conslutar el clima")
}
variable data = response.json
regresa data
```

Funcion remplazarDatos, recibe una referencia String con el nombre busqueda y una referencia JSON con el nombre climaData.

```
variable section <- document.querySelector('resultados')
section.innerHTML(etiquetaHTML.cargarClima(climaData,busqueda))
```

Funcion cargarDatosTicket()

```
variable fecha <- new Date()
variable form <- document.querySelector('.form')
form.addEventListener('sumbit', sub funcion(event){
  event.preventDefault()
  variable ticket <- document.querySelecto('#ticket').value
  variable url <- consulta coleccion ticket de mongo con variables ticket y fecha.getTime()
  Intentar{
    variable climaDAta <- metodo cargarClima que recibe la variable url
    metodo remplazarDatos que recibe la variable climaData y string "ticket"
  }
  Capturar{
    window.alert("mensaje de errror")
  }
}).
```

Funcion cargarDatosCiudad.

```
variable form <- document.querySelector('.form')
form.addEventListener('sumbit', sub funcion(event){
  event.preventDefault()
  variable ciudad <- document.querySelecto('#ciudad').value
  variable url <- consulta coleccion ciudad de mongo con variables ciudad y fecha.getTime()
  climaData <- metodo cargarClima que recibe la variable url
  ClimaData.then(funcion datos => { funcion remplazarDatosTicket que recibe variable datos, string
"ciudad")}).Error(async function(error producido de datos){
```

```
    console.log(error)
    window.alert("mensaje de error")
  })
})
```

Funcion cargarHTML(informacion del clima, busqueda)

```
html <--- 'Cargar contenido del nombre de la seccion';
Por cada elemento de climaData
  elemento <--- elemento de climaData;
  nombreCiudad <--- el nombre asociado al elemento
  html <-- encabezadoCiudad(Titulo del encabezado, imagenClima, AnimacionAvion)
  html <-- cargarSeccion(clima del elemento) //Se realiza para datos generales,caracteristicas del
viento y otras caracteristicas.
regresa el valor de htlm
```

Funcion cargarSeccion(encabezado,informacion)

```
seccion <-- cargar nombre de seccion
por cada elemento de informacion
  seccion <-- cargarIcono(elemento) //Metodo que carga un icono en html
regresa el valor de seccion;
```

Pseudocodigo para metodos de modelo.

Clase conexion:

Método insertar, recibe cliente de mongo, referencia en string de base de datos referencia a coleccion en string y un JSON llamado nuevo listado.

```
 constante resultado <- metodo insertOne de mongo
 regresar resultado
```

Método busquedadeconsultaBD recibe cliente de mongo, referencia en string de base de datos referencia a coleccion en string y un JSON llamado busqueda

```
constante resultado <- metodo find de mongo con la busqueda
constante doc[]
Para cada constate indice de resultado{
  agrega indice a doc
}
regresa doc
```

Método insertaclima, recibe referencia en string de base de datos, referencia a coleccion en string, referencia en string a un codigo IATA y un JSON llamado nuevaL(el forecast de openweather).

```
constante uri<- uri de la base de datos de mongoDB
constante cliente <- nuevo cliente de mongo
variable registro <- json que contiene IATA y un sub json de nombre clima
para cada elemento i=0; i<longitud de nuevaL; i++{
  variable creador <- un json con la informacion requerida sacada de nueva list(dt, main,
```

```
temperatura,presion,humedad,estado,descripccion,viento,velocidad,direccion,fuerza,visibilidad,preci
pitacion,lluvia,nieve,fecha)
  Si nuevaL.list[i].rain !== undefined{
    creador.lluvia<-nuevaL.list[i].rain en  3 horas
  }
  Si nuevaL.list[i].snow !== undefined{
    creador.nieve<-nuevaL.list[i].snow en  3 horas
  }
  registro.clima[nuevaL.list[i].dt]<- creador
}
variable insertado <- false
Intentar{
  esperar conexion de cliente
  insertado <- insertar(cliente,baseDatos,coleccion,registro)
} Capturar{
  error
}Finalmente{
  espera cerrar la conexion con el cliente
}
regresa el insertado.
```

Método consultaBD, recibe referencia en string de base de datos, referencia a coleccion en string y un JSON llamado busqueda.

```
constante uri <- uri de la abse de datos.
costante cliente <- nuevo cliente de mongo
variable consulta []
Intentar{
  espera conexion de cliente
  consulta <- busquedadeconsultaBD(cliente,base de datos, coleccion, busqueda).
} Capturar{error}
Finalmente{
  espera cierre de conexion del cliente
}
regresa consulta.
```

Método vacia, recibe referencia en string de base de datos, una referencia a coleccion en string y un json que sera el filtro.

```
constante uri <- uri de la abse de datos.
costante cliente <- nuevo cliente de mongo
variable eliminar <- false
Intentar{
  espera conexion de cliente
  variable eliminado <- metodo deleteMany de mongo con el filtro
  eliminar <- verificar que eliminado sea true
} Capturar{error}
Finalmente{
  espera cierre de conexion del cliente
}
regresa elminar.
```

Método consultaClima, recibe referencia en string de base de datos, referencia a coleccion en string , un JSON llamado busqueda y la fecha unix en string.

```
constante uri <- uri de la base de datos de mongo
constante cliente <- nuevo cliente de mongo
variable consulta <- []
Intentar{
  espera conecion de cliente
  constante consultado <- busquedadeconsultaBD(cliente,base de datos, coleccion, busqueda)
  Si la longitud de consultado es diferente de 0{
    Para cada elementeo i=0 hasta i<longitud de temp i++
      variable clima <- consultado[i].clima[fechaUnix]
      agrega a consulta ({
        objeto json con IATA <- i-esimo elemento de consultado con el valor del IATA
        clima <- variable clima})
}Capturar{error}
Finalmente{
  espera cierre de conecion de cliente
}
regresa consulta.
```

Método insetarVariosBD, recibe referencia en string de base de datos, una referencia a coleccion en string y un json que sera el nuevoListado.

```
constante uri <- uri de la base de datos de mongo
costante cliente <- nuevo cliente de mongo
variable insertado <- false
Intentar{
  espera conexion de cliente
  insertado<- metodo inserMany de mongo
}Capturar{error
}Finalmente{
espera cierre de conexion de cliente
}
regresa insertado.
```

Clase actualizabd:

Método obtenerTicketsCiudades, recibe dos JSON llamados datosCSV y ciudades.

```
variable ticketarreglo = []
Para cada ellemento i=0 hasta i<longitud de datosCSV; i++{
  variable tickets = {}
  constante elemento <- datosCSV[i]
  tickets = elemento json que contiene ticket <- num_ticket del elemento, ciudad_origen <- origien
del elemento, ciudad destino <- destino del elemento.
  agregar tickets en ticketArreglo
  Si ciudades[0][elemento.origin]== undefined {
    ciudades[0][elemento.origin] <- elemento json con ciudad teniendo una string vacia, cordenadas
latitud, longitud y IATA extraidos del elemento.origin
    agregar elemento.origin a ciudades[1]["ciudades"]
  }
  Si ciudades[0][elemento.destination]== undefined {
```

```
    ciudades[0][elemento.destination] <- elemento json con ciudad teniendo una string vacia,
cordenadas latitud, longitud y IATA extraidos del elemento.destinaton
    agregar elemento.destination a ciudades[1]["ciudades"]
  }
}
regresa un objeto json con ticketsAlta <- ticketArreglo, ciudadesAlta <- ciudades.
```

Método agregaInformacionCSV, recibe una referencia string llamada direccionCSV.

```
variable csvToJson <- requiere el paquete 'convert csv to json'
constante conexion <- requiere clase conexion
constante datosCSV <- csvToJson.metodos de converttojson para direccionCSV
variable ciudades <-metodo consultaBD de conexion con la base de datos de mongo, la coleccion
ciudad de mongo y un json vacio{}
metodo de conexion vacia con la base de datos de mongo, la coleccion ciudad de mongo y un json
vacio{}
Si ciudades[0] == undefined{
  ciudades[0]= {}
}
Si ciudades[1] == undefined{
  ciudades[1]= {"ciudades":[]}
}
variable resultadp <- obtenerTicketsCiudades(datosCSV,ciudades)
conexion.insertarVariosBD(base de datos de mongo, coleccion ticket de mongo, ticketArreglo).
conexion.insertarVariosBD(base de datos de mongo, coleccion ciudad de mongo, ciudades).
regresar resultado.
```

Método actualizaclimabd, recibe una referencia String con nombre guardarinfo

```
constante conexion <- requiere clase conecion
variable fs <- requiere paquete fs
variable fecha <- new Date()
variable recuclim <- conexion.consultaBD(base de datos mongo,clima mongo,{})
Si longitud de recuclim != 0{
  fs.appendFile(guardainfo +"./climaBD"+fecha.getTime()+".json", JSON.stringify(recuclim), async
function (err) {
    if (err) throw err)
    metodo vacia de conexion con la base de datos de mongo, la coleccion ciudad de mongo, {}
}
variable climas <- []
variable ciudades <- conexion.consultaBD con base de datos de mongo, coleccion ciudad de mongo, {}.
Si longitud de ciudades == 0{
  ciudades <- [{},{"ciudades":[]}]
}
Para cada elemento i=0 hasta i< longitud de ciudades[1]["ciudades"] i++{
  constante referencia <-ciudades[1]["ciudades"][i];
  constante ciudad <-ciudades[0][referencia];;
  constante clima <- metodo realizaPeticion con ciudad;
  Si clima != undefined{
    variable verificadoInsetar <- metodo instertar clima de conexion con base de datos de mongo,
coleccion clima de mongo,clima,ciudad.IATA
    Si verificadoInsertar{
```

```
      agrega clima a climas
    }
  }
  metodo petateate(4)
}
regresa climas.
```

-**Método petateate**, recibe un entero que sera la duracion de dormir en segundos el programa.

```
regresa una nueva promesa que dado el metodo timeout le dara la duracion del entero*1000
milisegundos para que siempre sean segundos.
```

-**Método realizaPeticion,** recibe un JSON con la informacion de la ciudad a consultar.

```
variable url <- llamada de la api de openwether con la latitud y longitud de la ciudad y la apikey
variable respuesta <- metodo fetch de url
Si respuesta.status != 200{
  regresa undefined
} en otro caso {
  regresa  el metodo.json de la respuesta.
}
```

Pesudocodigo para métodos de controlador:

Método actualizarBaseDeDatos, actualiza la base de datos de climas llamando a los metodos de la clase actualizabd

```
actualizarBaseDeDatos() {
    Intenta {
        climasActualizados <- espera actualizaBD.actualizaclimabd(archivo a guardar);
        Para (i<-0; i < logitud de climasActualizados; i++){
            clima = climasActualizados[i]
            if(clima.error != undefined){
                error del clima
            }
        }
    } atrapa (error) {
        Error al actualizar la base de datos
    }
}
```

Método actualizaTickets, actualiza la base de datos de tickets llamando a los metodos de la clase actualizabd.

```
actualizaTickets() {
    Intenta {
        direccionCSV <- direccion del archivo CSV
        ticketsActualizados <- espera actualizaBD.agregaInformacionCSV(direccionCSV)
    } atrapa (error) {
        Error al actualizar la base de datos
    }
}
```

Método obtenerClimaPorTicket, obtiene el clima de las ciudades de origen y destino de un ticket aceptando como parametros al request, response y la base de datos y regresa un json con el clima

```
obtenerClimaPorTicket(req, res,base_datos) {
    ticket <- req.query.ticket
    date <- req.query.date

    ticketData <- espera conexion.consultaBD(base_datos, coleccion de ticket, {"ticket" : ticket})
    ciudades <- espera conexion.consultaBD(base_datos, coleccion de ciudad, {})

    ciudad_origen <- ticketData[0].ciudad_origen
    ciudad_destino <- ticketData[0].ciudad_destino
    fecha <- nueva Date(date)
    consulta <- nueva Fecha con hora multiplo de 3 a partir de fecha

    let fechaUnix <- (''+consulta.getTime()).substring(0,10)

    const climas <- espera conexion.consultaClima(base_datos, coleccion de clima,
        {$or : [{"IATA" : ciudad_origen}, {"IATA" : ciudad_destino}]},fechaUnix)


    respuesta <- objeto JSON con los climas y busqueda

    regresa res.status(200).json(respuesta);
}
```

Método obtenerClimaPorCiudad, obtiene el clima de una ciudad especifica aceptando como parametros al request, response y la base de datos y regresa un json con el clima

```
obtenerClimaPorCiudad(req, res,base_datos) {
    const ciudad <- req.query.ciudad
    const date <- req.query.date

    const ciudades <- espera conexion.consultaBD(base_datos, process.env.coleccion_ciudad, {})

    let mejorCoincidencia <- ""
    let distanciaMinima <- Number.MAX_VALUE
    let ciudadIATA <- ""

    for(let i = 0; i < ciudades[1].ciudades.length; i++){
        elemento <- ciudades[1].ciudades[i]

        if(elemento == ciudad){
            mejorCoincidencia <- ciudades[0][elemento].ciudad
            ciudadIATA <- ciudades[0][elemento].IATA
            break;
        }

        distancia <- levenshtein.get(ciudad.toLowerCase(),
ciudades[0][elemento].ciudad.toLowerCase());
        if (distancia == distanciaMinima && distancia < 4) {
            distanciaMinima <- distancia
```

```
            mejorCoincidencia <- ciudades[0][elemento].ciudad
            ciudadIATA <- ciudades[0][elemento].IATA
        }
    }


    fecha = nueva Date(date);
    consulta <- nueva Fecha con hora multiplo de 3 a partir de fecha

    fechaUnix <- (''+consulta.getTime()).substring(0,10);

    climas <- espera conexion.consultaClima(base_datos, coleccion clima,
        {"IATA" : ciudadIATA},fechaUnix);

    respuesta <- objeto JSON con los climas y busqueda

    regresa res.status(200).json(respuesta);
}
```

Manuales y apoyo para la creacion del programa

Se utilizo los manuales de cada dependencia de NPM ya que se daba una breve explicacion de como usar el código y de esta manera se nos facilito usarlo. A su vez, para el código de MongoDB, ExpressJS y Jasmine se utilizo los manuales que se encuentran en sus páginas oficiales. En general se ocupo lo siguiente:

- Manual de dotenv de NPM [motdotla, 2023]
- Manual de NPM de jasmine para su instalacion. [jas, 2023]
- Manual de Jasmine para el entendimiento del paquete. [Jasmine, 2023]
- Manual de NodeJS para su instalacion. [nod, 2023]

Ayuda de modelo:

Para el modelo se ocupo lo siguiente:

- Manual de NPM de mongodb. [MongoDB, 2023]
- Manual de NPM de convert-csv-to-json [iuccio, 2023]
- Documentacion de Mongodb [Mon, 2023]
- Articulo para hacer el método sleep en javascript [Alarcon Jose Manuel, sf]

Ayuda del controlador

Para el controlador se utilizo lo siguiente:

- Manula de NPM de expressJS [expressjs, 2023]
- Documentacion de ExpressJS [Exp, 2023]
- Manual de NPM para fast-levenstein [hiddentao, 2023]
- Manual de NPM de node-cron [merencia, 2023]

Ayuda de la vista

Para la vista se utilizo lo siguiente:

- Manual de MDN para seleccion de elementos del DOM [mdn, 2023b]
- Manual de MDN para modificar elementos [mdn, 2023a]
- Plantilla de Selecao de Bootstrap para la parte visual, con varias modificaciones. [BootstrapMade,s.f]

Referencias

- [jas, 2023] (2023). DOCS. Jasmine. Recuperado de https://jasmine.github.io/pages/docshome.html.
- [mdn, 2023a] (2023a). Element. MDN. Recuperado de https://developer.mozilla.org/es/docs/Web/API/Element.
- [Exp, 2023] (2023). Guia. ExpressJs. Recuperado de https://expressjs.com/es/guide/routing.html.
- [mdn, 2023b] (2023b). Localizando elementos DOM usando selectores. MDN. Recuperado de https://developer.mozilla.org/es/docs/Web/API/Documentobjectmodel/LocatingDOMelementsusingselectors.
- [Mon, 2023] (2023). MongoDB Node Driver. MongoDB. Recuperado de https://www.mongodb.com/docs/drivers/node/current/.
- [nod, 2023] (2023). Node.js. Node.js. Recuperado de https://nodejs.org/es/.
- [Alarcon Jose Manuel, sf] Alarcon Jose Manuel (s.f). Como hacer un sleep() en Javas- cript: detener la ejecuci´on durante un tiempo. Infosecurity magazine. Recuperado de: https://www.campusmvp.es/recursos/post/como-hacer-un-sleep-en-javascript-detener-la-ejecucion- durante-un-tiempo.aspx.
- [expressjs, 2023] expressjs (2023). express. NPM. Recuperado de https://www.npmjs.com/package/express.
- [hiddentao, 2023] hiddentao (2023). fast-levenshtein. NPM. Recuperado de https://www.npmjs.com/package/fast-levenshtein.
- [iuccio, 2023] iuccio (2023). CSVtoJSON. NPM. Recuperado de https://www.npmjs.com/package/convert-csv-to-json.
- [merencia, 2023] merencia (2023). Node Cron. NPM. Recuperado de[https://www.npmjs.com/package/node-cron.
- [Jasmine, 2023] Jasmine (2023). The Jasmine Module. NPM. Recuperado de https://www.npmjs.com/package/jasmine.
- [MongoDB, 2023] MongoDB (2023). MongoDB Node.js Driver. NPM. Recuperado de https://www.npmjs.com/package/mongodb.

⑩ [motdotla, 2023] motdotla (2023). dotenv. NPM. Recuperado de https://www.npmjs.com/package/dotenv.

⑩ [BootstrapMade,s.f] BootstrapMade (s.f). Selecao - Agency Bootstrap Template. BootStrapMade. Recuperado de : https://bootstrapmade.com/selecao-bootstrap-template/