# UVM Forms Manual Técnico

Luis Monsalve, José Escalona, Maikel Villegas, Juan González

Entrega de Informe

Facultad de Ingeniería, Universidad “Valle del Momboy”

Front End I

Brian Santeliz

# Desarrollo

Diseño Propuesto: La Rest-API permite a los usuarios registrarse e iniciar sesión para crear, modificar y eliminar sus propios quizzes. También pueden responder a los quizzes creados por otros usuarios y ver las respuestas de otros usuarios a sus propios quizzes. Los usuarios pueden buscar quizzes por categoría o por palabra clave, así como obtener una lista de los quizzes más populares. La API utiliza Node.js y Express para manejar las solicitudes HTTP entrantes y MongoDB para almacenar los datos. Los datos se presentan en un sitio web implementado con EJS y CSS para proporcionar una experiencia de usuario agradable. Además, la API cuenta con medidas de seguridad para proteger los datos de los usuarios y evitar la exposición de información confidencial con medidas de seguridad para proteger los datos de los usuarios y evitar la exposición de información confidencial.

# Creación de la Base de Datos

La base de datos "UVM-Forms" almacena información relacionada con formularios y encuestas.

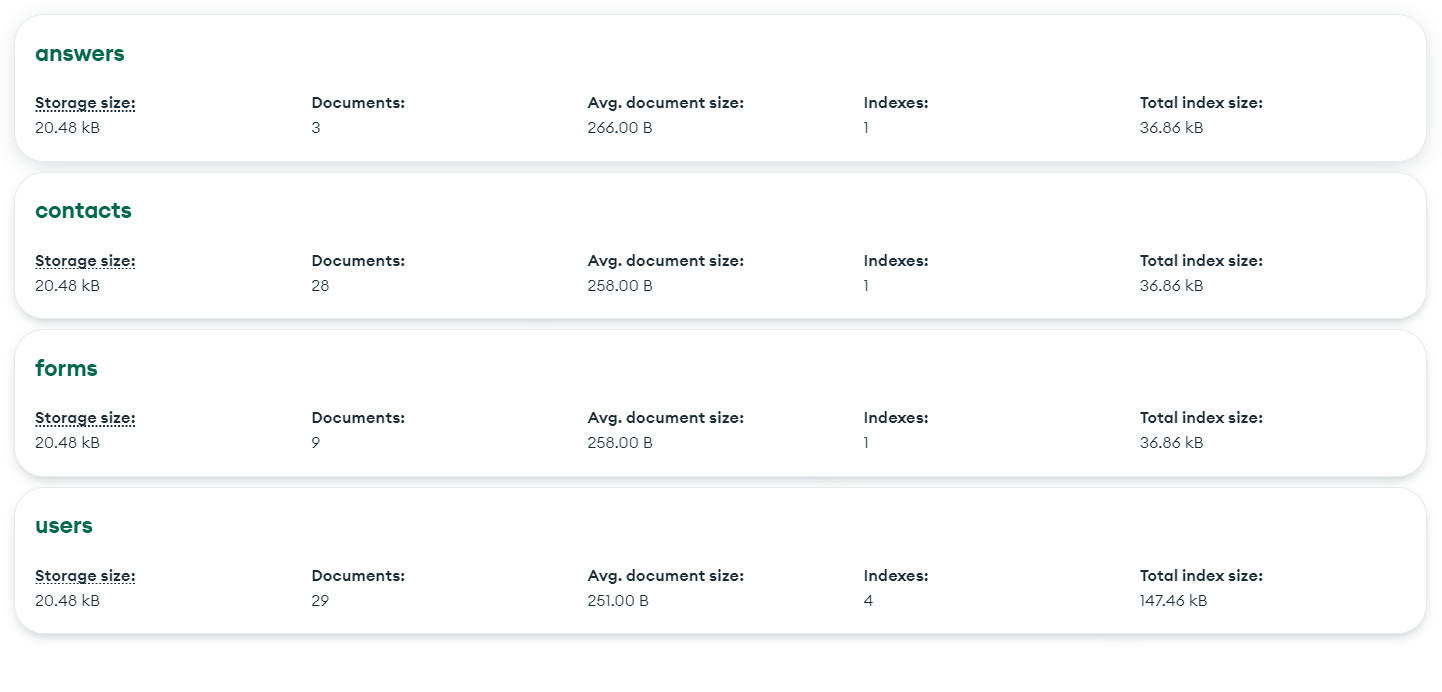
La colección "answers" de la base de datos "UVM-Forms" almacena las respuestas de los estudiantes a las encuestas y formularios. Cada respuesta se identifica de forma única mediante un campo "\_id". Además, cada respuesta está asociada a un formulario específico mediante un campo "id\_encuesta" y a un estudiante específico mediante un campo "id\_User". Las respuestas a cada campo del formulario se almacenan en un array llamado "respuestas". El campo "respuestas" contiene un objeto por cada campo del formulario, y cada objeto contiene la respuesta del estudiante a ese campo en particular. Cada objeto de respuesta contiene información como el tipo de campo (por ejemplo, texto, opción múltiple, selección única), el valor de la respuesta y cualquier otra información relevante para ese tipo de campo. La colección "answers" también contiene campos de fecha de creación y modificación para cada respuesta.

La colección "forms" de la base de datos "UVM-Forms" almacena información sobre los formularios y encuestas creados por los usuarios. Cada formulario se identifica de forma única mediante un campo "\_id" y contiene información como el título, la descripción y el autor del formulario. Los campos del formulario se almacenan en un array llamado "preguntas". Cada pregunta del formulario contiene información como el tipo de campo, el texto de la pregunta y cualquier otra información relevante para ese tipo de campo. La colección "forms" también contiene campos de fecha de creación y modificación para cada formulario.

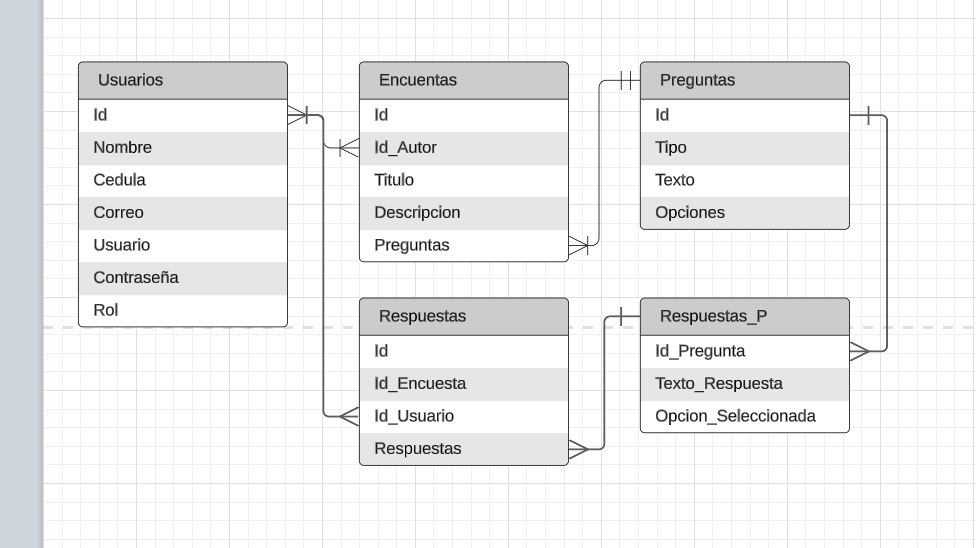
La colección "usuarios" de la base de datos "UVM-Forms" almacena información sobre los usuarios del sistema. Cada usuario se identifica de forma única mediante un campo "\_id" y contiene información como el nombre, la cédula, el correo electrónico, el nombre de usuario y la contraseña del usuario. El campo "rol" indica el rol del usuario en el sistema (por ejemplo, "administrador" o "usuario"). La colección "usuarios" también contiene campos de fecha de creación y modificación para cada usuario.

La colección “contacts” de la base de datos de “UVM-Forms” almacena información sobre las solicitudes enviadas por los usuarios desde la pagina de “Contacto” que sirve para contactar con el usuario. En donde, cada usuario tendrá que llenar los siguientes datos: Nombre Completo, Email, Cedula/ID, Fecha de Nacimiento, Genero, Número Telefónico, Asunto, Descripcion del Asunto y una imagen

***Base de Datos***

******

***Diagrama propuesto para la creación de la BD:***Este diagrama es una versión preliminar que nos sirvió para la elaboración de la base de datos de nuestro sistema



# Dependencias de Desarrollo

bcryptjs: Una librería para el hashing de contraseñas.

connect-flash: Un middleware de Express para mostrar mensajes flash en la aplicación.

cookie-parser: Un middleware de Express para manejar cookies en la aplicación.

ejs: Un motor de plantillas para generar HTML dinámico en el servidor.

express: Un framework de Node.js para construir aplicaciones web.

express-handlebars: Un motor de plantillas para generar HTML dinámico en el servidor con soporte para Handlebars.

express-session: Un middleware de Express para manejar sesiones en la aplicación.

jsonwebtoken: Una librería para la autenticación basada en tokens JWT.

method-override: Un middleware de Express para habilitar HTTP PUT y DELETE en navegadores que no los soportan.

mongoose: Una librería para interactuar con bases de datos MongoDB de manera más fácil y con menos código.

passport: Un framework de autenticación para Node.js.

passport-local: Una estrategia de autenticación para Passport que utiliza nombre de usuario y contraseña.

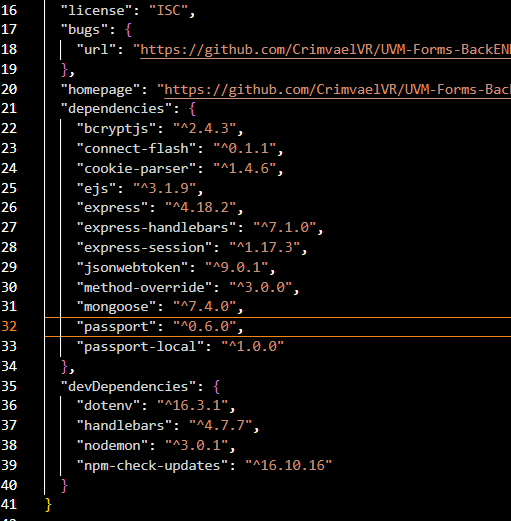
En cuanto a las dependencias de desarrollo:

dotenv: Una librería para cargar variables de entorno desde un archivo .env.

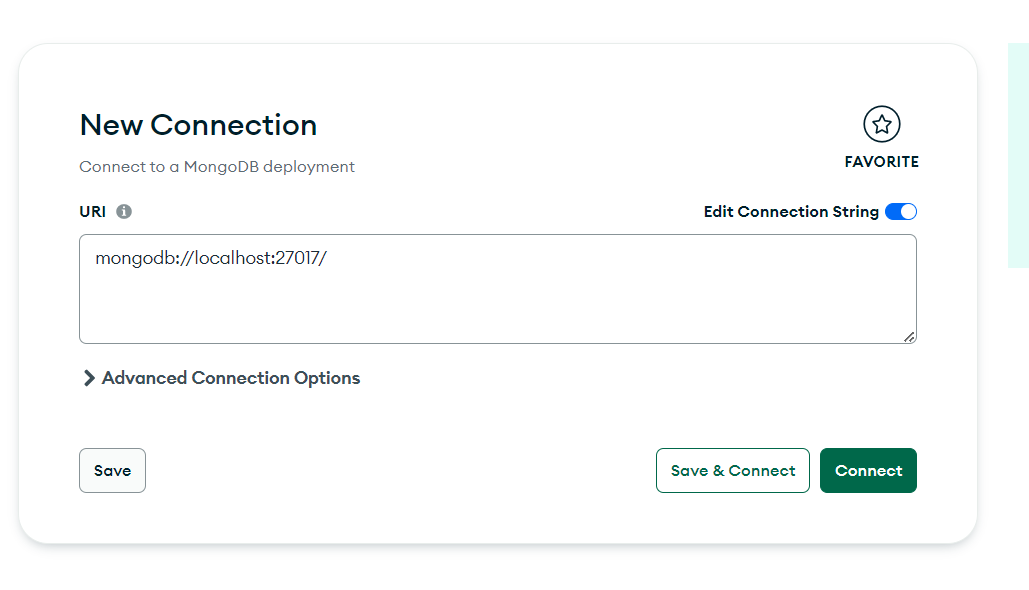
handlebars: Un motor de plantillas para generar HTML dinámico en el servidor con soporte para Handlebars.

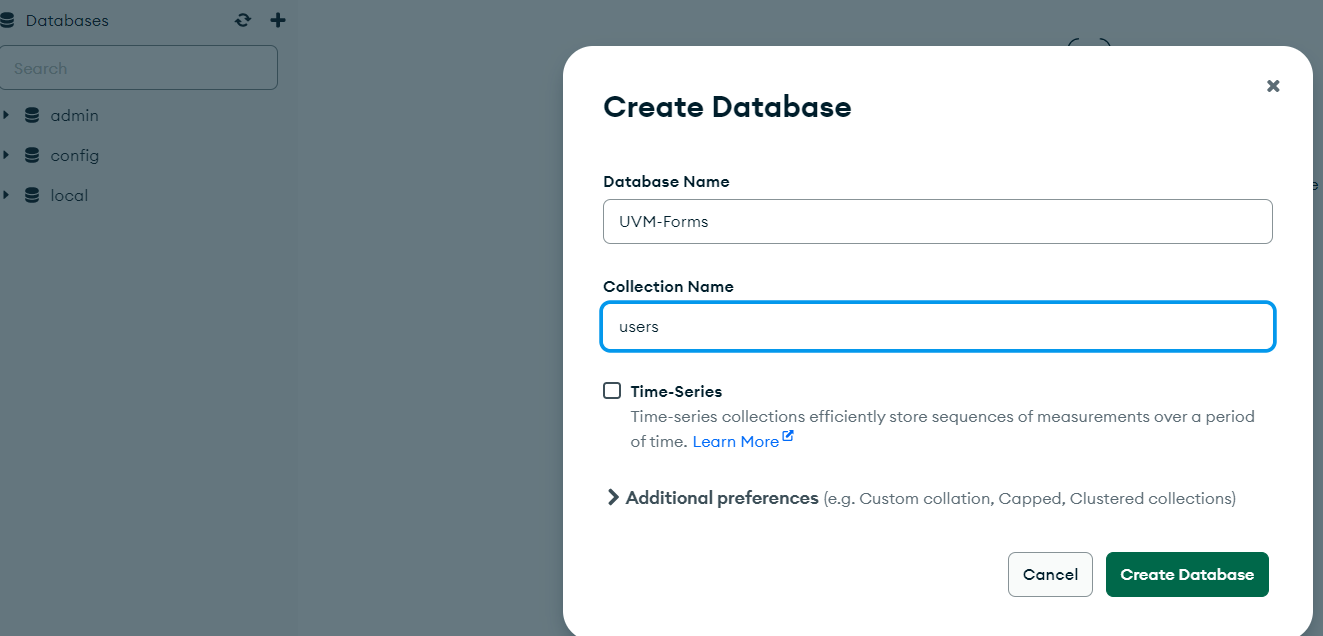
nodemon: Una herramienta que reinicia automáticamente la aplicación cuando se detectan cambios en el código fuente.

npm-check-updates: Una herramienta para actualizar las dependencias del proyecto a sus últimas versiones compatibles.

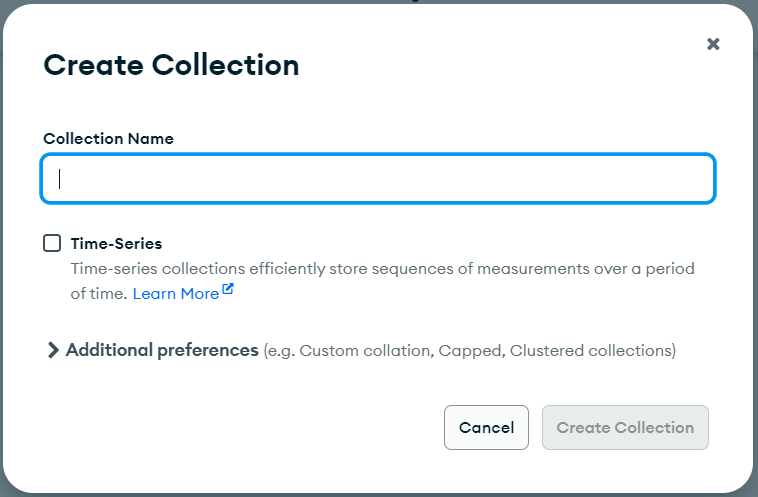


# Instalación

1. Antes de instalar requerirás los siguientes programas y componentes: Visual Studio Code, Node JS, MongoDB, MongoDB Server y MongoDB Compass
2. Lo primero es descargar ambos repositorios: <https://github.com/CrimvaelVR/UVM-Forms-FrontEND> (El front) y <https://github.com/CrimvaelVR/UVM-Forms-BackEND>
3. Luego de descargarlos vamos a pasar el contenido de la carpeta de Front-End al src del Back-End
4. Lo siguiente es hacer la DB a partir de MongoDB Compass, vamos a abrir el programa y conectaremos hacia la siguiente dirección: 
5. Crearemos una base de datos desde el signo de suma de arriba a la izquierda y asignaremos el nombre de la base de datos que necesitamos(UVM-Forms) y la creación de una de las colecciones necesarias



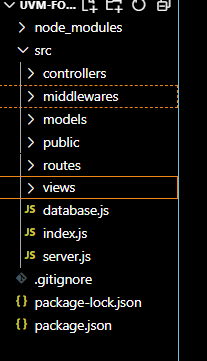
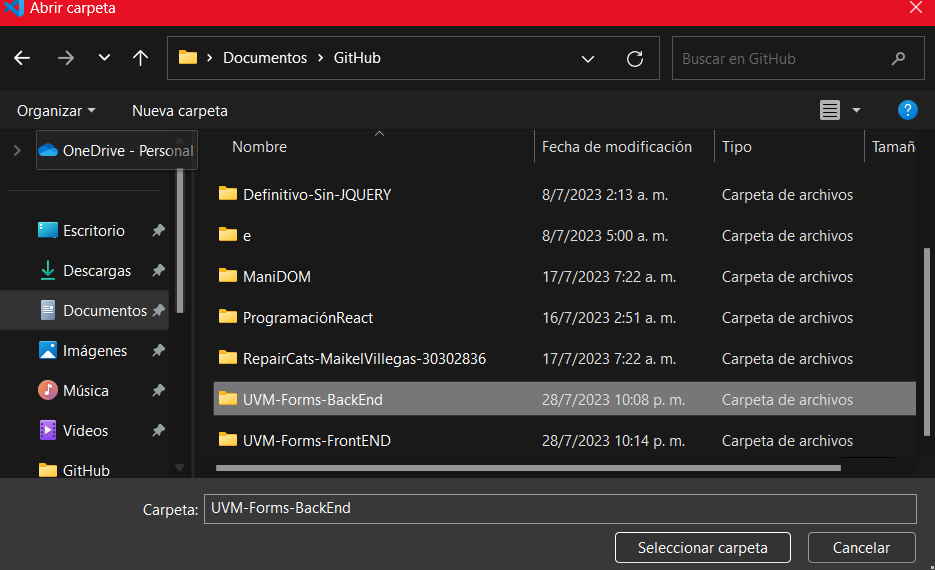
1. Crearemos las otras tres colecciones(answers, contacts y forms) necesarias con el signo de agregar que se encuentra al lado de la BD



1. Nos debería quedar como fue mencionado anteriormente en el documento
2. Ahora, abrimos Visual Studio Code y seleccionamos en Archivo/Abrir Carpeta, luego de

esto seleccionan la carpeta de App\_Backend, de esta manera podrán ver el código del

proyecto y podrán inicializarlo

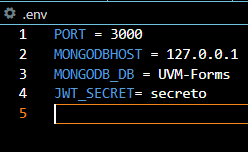


1. Como se percatarán en la carpeta descargada no se encuentra el archivo “. env”, eso es

porque debemos crearlo para poder acceder a la base de datos en nuestro servidor, para

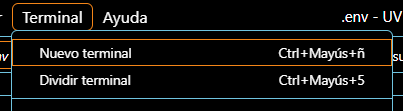
eso simplemente creamos un archivo llamado “. env” (se puede realizar con el Icono de

Archivo+), y colocaremos las siguientes variables, también llamadas variables de entorno

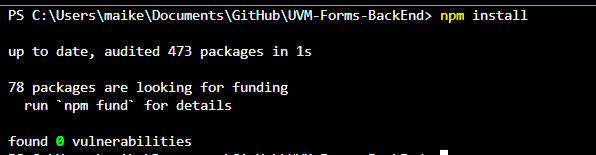


1. Lo siguiente es crear una terminal desde Visual Studio Code, con las teclas

CTRL+Mayus+Ñ o desde Terminal/Nuevo Terminal



1. Antes de inicializar el servidor, colocaremos un npm install en la terminal para instalar todas las dependencias



1. Ahora usamos npm run dev y si tenemos todo correcto nos saldrá lo siguiente



1. Ahora, por ejemplo, si vamos a localhost:3000 podremos ver la landing page

# Explicaciones Generales

El archivo server.js es el archivo principal de la aplicación. En este archivo se importan los módulos necesarios, se configuran las opciones de la aplicación y se definen las rutas para los diferentes recursos. También se agregan algunos middlewares y se configura el manejo de archivos estáticos. Finalmente, se exporta la aplicación.

El archivo index.js es el punto de entrada de la aplicación. En este archivo se importa la configuración de la aplicación y se conecta a la base de datos. Luego se inicia el servidor y se escucha en el puerto especificado.

El archivo database.js se encarga de establecer la conexión con la base de datos MongoDB. Se utiliza la URL de conexión proporcionada en las variables de entorno y se configuran algunas opciones adicionales. Si la conexión es exitosa, se muestra un mensaje de "Base de Datos Conectada". Si hay algún error, se muestra el mensaje de error.

El middleware de auth se utiliza para verificar y gestionar la autenticación y autorización de los usuarios en la aplicación.

La función verifyToken es un middleware que verifica si el token JWT proporcionado en la cookie de la solicitud es válido. Si el token es válido, se decodifica y se guarda la información del usuario en el objeto req. Si el token no es válido o no existe, se devuelve un error de acceso denegado.

La función verificarSesion es otro middleware que verifica si el usuario está logueado. Verifica si el token JWT proporcionado en la cookie de la solicitud es válido. Si el token es válido, se redirige al usuario a otra página, por ejemplo, el panel de usuario. Si el token no es válido o no existe, se continúa con el siguiente middleware.

El middleware de bcrypt se utiliza para encriptar contraseñas y comparar contraseñas encriptadas con contraseñas sin encriptar.

La función encriptarContrasena es una función asincrónica que utiliza el algoritmo de hashing bcrypt para generar un hash de la contraseña proporcionada.

El middleware encriptarContrasenaMiddleware se utiliza para encriptar la contraseña antes de guardarla en la base de datos. Si no se proporciona una contraseña en la solicitud, se pasa al siguiente middleware sin hacer nada. Si se proporciona una contraseña, se genera un hash utilizando la función encriptarContrasena y se reemplaza la contraseña original con el hash en el objeto req.

La función compare se utiliza para comparar una contraseña sin encriptar con una contraseña encriptada utilizando bcrypt.

El middleware verifyrole se utiliza para verificar el rol del usuario y permitir o denegar el acceso a ciertas rutas según los roles especificados. Verifica el token JWT proporcionado en la cookie de la solicitud y obtiene el nombre de usuario del token. Luego busca el usuario en la base de datos y obtiene su rol. Si el rol del usuario coincide con alguno de los roles especificados, se permite el acceso a la ruta. Si el rol no coincide o el usuario no existe, se devuelve un error de permiso denegado o acceso denegado.

El script de buscarform.js, se utiliza para asignar una URL a un enlace de búsqueda cuando se hace clic en él o se presiona la tecla Enter en un campo de texto.

En este caso, el script busca los elementos del DOM con los ID 'buscar', 'textoBuscar' y 'formBuscar'. Luego, define la función asignarURL, que se ejecuta cuando se hace clic en el elemento con ID 'buscar'. Dentro de esta función, se obtiene el valor del campo de texto con ID 'textoBuscar' y se verifica si está vacío. Si está vacío, se modifican las clases y el marcador de posición del elemento con ID 'formBuscar' para mostrar un mensaje de error. Si no está vacío, se asigna una URL al enlace con ID 'buscar' concatenando el valor del campo de texto y se llama a la función showLoader.

El script de loader.js, se utiliza para mostrar y ocultar un elemento de carga mientras se carga la página.

En este caso, el script define las funciones showLoader y hideLoader, que modifican la visibilidad y opacidad de un elemento con ID 'contenedor\_carga' para mostrar u ocultar el elemento de carga respectivamente. Además, el script utiliza el evento onload para ocultar el elemento de carga cuando la página se ha cargado completamente. También utiliza el evento pageshow para ocultar el elemento de carga si la página se ha almacenado en caché y se vuelve a cargar.

Patron MCR: El patrón Modelo-Controlador-Rutas (MCR) es una arquitectura comúnmente utilizada en aplicaciones web desarrolladas con Express, Node.js y MongoDB. Este patrón divide la aplicación en tres componentes principales:

1. Modelo: El modelo se encarga de interactuar con la base de datos (en este caso, MongoDB). Define la estructura y las operaciones que se pueden realizar en los datos. Por ejemplo, puede definir un modelo de usuario que tenga propiedades como nombre, correo electrónico y contraseña, y métodos para crear, actualizar, eliminar y buscar usuarios en la base de datos.
2. Controlador: El controlador es responsable de manejar las solicitudes HTTP y realizar la lógica de negocio de la aplicación. Recibe las solicitudes del cliente a través de las rutas y utiliza los modelos correspondientes para interactuar con la base de datos.
3. Rutas: Las rutas definen las URL y los métodos HTTP que se pueden utilizar para acceder a diferentes recursos de la aplicación. Estas rutas están asociadas a los controladores correspondientes que manejan las solicitudes. Por ejemplo, puede haber una ruta "/usuarios" que acepte solicitudes GET para obtener una lista de usuarios y solicitudes POST para crear un nuevo usuario.

El patrón MCR ayuda a mantener una estructura organizada y modular en una aplicación web. Separar la lógica de negocio en controladores y modelos facilita la reutilización del código y permite realizar pruebas unitarias más fácilmente. Además, las rutas proporcionan una forma clara de definir las API de la aplicación y cómo se deben manejar las solicitudes entrantes. En general, el patrón MCR mejora la mantenibilidad y escalabilidad de una aplicación web.

La carpeta "public" y sus subcarpetas "resources", "css" y "scripts" se utilizan para almacenar archivos estáticos que serán accesibles públicamente desde el navegador.

La carpeta "resources" se utiliza para almacenar recursos como imágenes, fuentes, archivos de audio, etc., que se utilizarán en la página web.

La carpeta "css" se utiliza para almacenar archivos CSS que contienen estilos que se aplicarán a la página web.

La carpeta "scripts" se utiliza para almacenar archivos JavaScript que contienen código que se ejecutará en la página web.

La carpeta "views" se utiliza para almacenar archivos HTML o plantillas que definen la estructura y el contenido de las páginas web. Estos archivos se utilizan para generar las vistas dinámicas de la aplicación web.