BOTBLOQ: Ecosistema integral para el diseño, fabricación y programación de robots DIY

Proyecto Financiado por el Centro de Desarrollo Tecnológico Industrial (CDTI)
EXPEDIENTE: IDI-20150289
Cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Operativo Plurirregional de Crecimiento Inteligente 2014-2020

ACRÓNIMO DEL PROYECTO: BOTBLOQ

MINISTERIO DE ECONOMÍA Y COMPETITIVIDAD

CDTI Centro para el Desarrollo Tecnológico Industrial

UNIÓN EUROPEA
Fondo Europeo de Desarrollo Regional (FEDER)
Una manera de hacer Europa

ENTREGABLE E.6.1.2 Arquitectura Técnica

RESUMEN

Principales componentes de la arquitectura técnica del sistema: arquitectura REST, Node.js, Express, MongoDB, otras librerías auxiliares y su funcionalidad, estructura de los principales componentes (routes, model, controller, functions).

Índice

Proyecto Financiado por el Centro de Desarrollo Tecnológico Industrial (CDTI).
Expediente IDI-20150289
Cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Operativo Plurirregional de Crecimiento Inteligente 2014-2020

Arquitectura REST (Representational State Transfer)

La arquitectura REST (Representational State Transfer) es una forma de proporcionar interoperabilidad entre sistemas informáticos en Internet que se utiliza con frecuencia en el desarrollo de servicios Web. Su uso es preferido porque no requiere tanto ancho de banda, lo que es una ventaja para su uso en Internet. REST es un estilo de arquitectura de software y no un conjunto de estándares como otros enfoques (SOAP, etc).

En un servicio web REST, las solicitudes realizadas a través de la URI de un recurso generan una respuesta en formato XML, HTML, JSON, etc., la cual puede confirmar que se ha hecho alguna alteración en el recurso almacenado y proporcionar enlaces de hipertexto a otros recursos. Mediante el uso de HTTP, el tipo de operaciones disponibles incluyen los predefinidos por los verbos HTTP GET, POST, PUT, DELETE, etc. Las aplicaciones REST utilizan solicitudes HTTP para publicar datos (crear y/o actualizar), leer datos y eliminar datos mediante las operaciones CRUD (Create/Read/Update/Delete).

Mediante el uso de un protocolo sin estado y de operaciones estándar, los sistemas REST permiten un elevado rendimiento y fiabilidad, así como gran capacidad de crecimiento, utilizando componentes reutilizados que se pueden gestionar y actualizar sin afectar al sistema en su conjunto, aun cuando se esté ejecutando.

Los servicios REST son:

- Independientes de la plataforma (Unix, Mac, etc.) y del lenguaje de programación (se puede comunicar Java con .NET, etc),
- Basados en estándares (se ejecuta sobre HTTP) y
- Se puede utilizar fácilmente en presencia de firewalls.
- El estado y la funcionalidad se dividen en recursos distribuidos
- Cada recurso es direccionable de manera exclusiva utilizando un conjunto de comandos uniforme y mínimo (como GET, POST, PUT o DELETE)
- El protocolo es cliente / servidor, cada solicitud se considera independiente de cualquier solicitud anterior, en capas y permite almacenamiento en caché

La lógica de diseño detrás de la arquitectura REST puede describirse mediante un estilo arquitectónico que consiste en un conjunto de restricciones aplicadas a los elementos dentro de la arquitectura.

- **Estilo cliente-servidor**: Al separar los requerimientos de la interfaz de usuario de los requerimientos de almacenamiento de datos, se mejora la portabilidad a través de múltiples plataformas y la escalabilidad simplificando los componentes del servidor. La separación permite que los

Proyecto Financiado por el Centro de Desarrollo Tecnológico Industrial (CDTI).
Expediente IDI-20150289
Cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Operativo Plurirregional de Crecimiento Inteligente 2014-2020

componentes evolucionen de forma independiente con mayor facilidad.

- **Comunicación sin estado**: cada solicitud del cliente al servidor debe contener toda la información necesaria para comprender la solicitud, sin utilizar ninguna otra información almacenada en el servidor. Por tanto, el estado de la sesión se mantiene enteramente en el cliente. Esta restricción induce las propiedades de visibilidad, fiabilidad y escalabilidad. Se mejora la visibilidad porque un sistema de monitoreo no tiene que mirar más allá de la petición para determinar la naturaleza completa de la solicitud. La confiabilidad mejora porque se facilita la tarea de recuperación de fallos parciales. La escalabilidad se mejora porque al no tener que almacenar el estado entre las solicitudes, se permite que el servidor libere rápidamente recursos y se simplifica aún más la implementación porque el servidor no tiene que administrar el uso de recursos entre las solicitudes.
- **Utilización del caché**: requiere que los datos dentro de una respuesta a una solicitud estén etiquetados implícita o explícitamente como almacenables (o no) en caché. Si una respuesta se puede almacenar en caché, entonces se ofrece a la caché del cliente el derecho de reutilizar los datos de respuesta para solicitudes equivalentes posteriores. La ventaja de agregar restricciones de caché es que tienen el potencial de eliminar parcial o totalmente algunas interacciones, mejorando la eficiencia, la escalabilidad y el rendimiento percibido por el usuario mediante la reducción de la latencia promedio de una serie de interacciones.
- **Interfaz uniforme entre componentes**: Al aplicar el principio de ingeniería de software de generalidad a la interfaz de los componentes, la arquitectura general del sistema se simplifica y se mejora la visibilidad de las interacciones. Las implementaciones están disociadas de los servicios que proporcionan, lo que fomenta la evolución independiente. La interfaz REST está especialmente diseñada para realizar la transferencia de datos hypermedia de forma eficiente, que es el caso más común de la Web.
- **Sistema en capas**: permite el uso de capas jerárquicas, las cuales restringen el comportamiento de los componentes, ya que cada componente no puede "ver" más allá de la capa inmediata con la que están interactuando. Al restringir el conocimiento del sistema a una sola capa, limitamos la complejidad del sistema global y se promueve la independencia del sustrato. Las capas se pueden utilizar para encapsular servicios heredados y para proteger nuevos servicios de clientes heredados, simplificando componentes moviendo la funcionalidad poco utilizada a un intermediario compartido. Los intermediarios también se pueden utilizar para mejorar la escalabilidad del sistema permitiendo el balance de carga de los servicios a través de múltiples redes y procesadores.
- **Código a petición**: REST permite ampliar la funcionalidad del cliente descargando y ejecutando código en forma de applets o scripts. Esto simplifica los clientes reduciendo el número de características que deben ser implementadas a priori. La extensibilidad del sistema mejora al permitir que las características se descarguen después del despliegue.

Proyecto Financiado por el Centro de Desarrollo Tecnológico Industrial (CDTI).
Expediente IDI-20150289
Cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Operativo Plurirregional de Crecimiento Inteligente 2014-2020

Las características arquitectónicas afectadas por las restricciones del estilo REST son:

- Rendimiento: Las interacciones entre los componentes pueden ser el factor dominante en el rendimiento percibido por el usuario y en la eficiencia de la red
- Escalabilidad para soportar un gran número de componentes e interacciones entre componentes.
- Simplicidad mediante una interfaz uniforme
- Modificabilidad de los componentes para satisfacer las necesidades cambiantes (incluso mientras la aplicación se está ejecutando)
- Visibilidad de la comunicación entre componentes por agentes de servicio
- Portabilidad de los componentes moviendo el código del programa con los datos
- La fiabilidad es la resistencia al fallo en el sistema en presencia de fallas dentro de componentes, conectores o datos

Proyecto Financiado por el Centro de Desarrollo Tecnológico Industrial (CDTI).
Expediente IDI-20150289
Cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Operativo Plurirregional de Crecimiento Inteligente 2014-2020

Node.js

Antiguamente, los usuarios de Internet se dedicaban leer, ver y descargar contenido. Para chatear se utilizaban los tableros de mensajes o blogs, no se hacía en tiempo real.

Pero, hoy en día los usuarios de Internet quieren ser capaces de recibir retroalimentación instantánea. Actualmente, chatear, los juegos y la colaboración se llevan a cabo en tiempo real unido a la masividad del intercambio con millones de usuarios que interactúan en un sitio a la vez.

Desafortunadamente, cuando se creó la Web no se previó esta situación. Tener respuesta en tiempo real significa que cada cliente estaría realizando pedidos constantemente al servidor, provocando que los tiempos de respuesta fuesen lentos e ineficientes.

Para lograr respuestas rápidas de la Web, las comunicaciones no se pueden iniciar desde el lado del cliente. Por el contrario, los servidores tienen que enviar los datos a los clientes, cuando dichos datos estén disponibles. De lo contrario, el proceso sería igualmente tedioso.

Node.js es un entorno de tiempo de ejecución de JavaScript de código abierto y multiplataforma que permite desarrollar una gran variedad de herramientas y aplicaciones. Aunque Node.js no es un framework de JavaScript, muchos de sus módulos básicos están escritos en JavaScript y los desarrolladores pueden escribir nuevos módulos en JavaScript. El entorno de tiempo de ejecución interpreta JavaScript utilizando el motor JavaScript de V8 diseñado por Google para Chrome. V8 traduce JavaScript a lenguaje de máquina nativo, en lugar de interpretarlo como código de bytes. Esto permite a Node.js crear un entorno en tiempo de ejecución que transfiere JavaScript desde el servidor al cliente rápidamente.

Node.js es un sistema en tiempo de ejecución para crear aplicaciones del lado del servidor, para que los codificadores de JavaScript construyan APIs en tiempo real.

Node.js permite la creación de servidores Web y herramientas de red utilizando JavaScript y una colección de módulos para E/S del sistema de archivos, redes (DNS, HTTP, TCP, TLS/SSL o UDP), datos binarios (buffers), funciones de criptografía, flujos de datos y otras funciones básicas. Los módulos de Node.js utilizan una API diseñada para reducir la complejidad de las aplicaciones de escritura de servidor.

Por todo lo antes mencionado, Node.js es un entorno muy útil en la construcción de aplicaciones en red rápidas y escalables. Esta velocidad, combinado con la programación asincrónica, son las características más destacables de Node.js

Proyecto Financiado por el Centro de Desarrollo Tecnológico Industrial (CDTI).
Expediente IDI-20150289
Cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Operativo Plurirregional de Crecimiento Inteligente 2014-2020

La comunicación humana sucede de forma caótica, sin la debida sincronía: uno no espera a que le respondan para hablar de nuevo, a diferencia de lo que ocurre con los lenguajes de programación. Node.js funciona de la misma manera asíncrona.

Node.js es un motor de servidor altamente personalizable que no hace nada mientras no se lo configura. Este prototipo de servidor procesa mediante un Ciclo de Eventos, encargado de aceptar y responder a las peticiones, asegurando que no se soliciten datos constantemente, sino que simplemente se trasmitan cuando existan. Así, mientras el Ciclo de Eventos se encarga del manejo de las tareas asíncronas, Nodo.js puede continuar ejecutando el programa normalmente y dejar de lado el trabajo pesado.

Cualquiera de estas peticiones puede iniciar otras peticiones a alguna otra parte del sistema, tales como leer un archivo fuera de disco o enviar una señal para girar un motor en un brazo de robot. Ese bucle, conocido como el bucle de eventos, es la parte "runtime".

Esto significa que los desarrolladores pueden usar Node.js para crear aplicaciones que escalen a millones de usuarios. La comunicación en tiempo real es gestionada por el Ciclo de Eventos sin ocupar mucho espacio en la memoria, por lo que los desarrolladores pueden dedicar más tiempo a trabajar en la funcionalidad de la aplicación sin tener que preocuparse por que la aplicación se atasque con demasiadas consultas.

Node.js tiene una arquitectura basada en eventos capaz de realizar E/S asíncronas. Estas opciones de diseño apuntan a optimizar el rendimiento y la escalabilidad en aplicaciones Web con muchas operaciones de E/S, así como para aplicaciones Web en tiempo real (por ejemplo, programas de comunicación en tiempo real y juegos de navegador).

Node.js se utiliza principalmente para construir programas de red como servidores Web, haciéndolo similar a PHP. La mayor diferencia entre Node.js y PHP es que la mayoría de las funciones en PHP bloquean hasta la finalización, mientras que las funciones en Node.js están diseñadas para no bloquear (los comandos se ejecutan en paralelo y usan Callbacks a la conclusión de la señal o en caso de falla).

Las aplicaciones Node.js pueden ejecutarse en servidores Mac OS X, Microsoft Windows, NonStop y Unix.

Proyecto Financiado por el Centro de Desarrollo Tecnológico Industrial (CDTI).
Expediente IDI-20150289
Cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Operativo Plurirregional de Crecimiento Inteligente 2014-2020

Express.js

Express.js y Node.js proporcionan a JavaScript una funcionalidad de back-end que permite a los desarrolladores crear software con JavaScript en el servidor, haciendo posible construir un sitio completo con JavaScript.

El entorno de Express permite integrar la estructura y las funciones necesarias para construir un sitio, para lo cual no estaba destinado Node. Es un marco bastante ligero que permite a los desarrolladores funciones adicionales de aplicaciones web integradas y la API Express, sin necesidad de sobrescribir la plataforma Node.

Los aspectos básicos del entorno Express son:

- Entorno para desarrollar aplicaciones web del lado del servidor y para móviles
- Lenguaje: escrito en JavaScript
- Plataforma: Node.js
- Express permite construir:
    - Aplicaciones web y móviles con una o más páginas e híbridas
    - Funciones comunes de back-end para aplicaciones web
    - API (interfaces de programación de aplicaciones)
- Motores de modelado: Express viene con dos motores de modelado, Jade y EJS, que facilitan el flujo de datos en una estructura de sitio web.
- Modelo MVC: Express soporta la arquitectura MVC (Model-View-Controller), muy útil para crear sitios web en un formato orientado por modelos.
- Sistema operativo: Es multiplataforma, por lo que no se está limitado a un solo sistema operativo.
- El Generador Express permite crear aplicaciones complejas rápidamente.

Express corresponde a la "E" de la pila de software MEAN, donde se ejecuta junto con la base de datos MongoDB, el entorno Node.js y el entorno front-end AngularJS.

A diferencia de otras pilas de software tradicionales, la pila MEAN está completamente basada en JavaScript, gracias a Node.js, el entorno de desarrollo JS que proporciona funcionalidad de back-end a JavaScript.

Express es un marco de aplicación web para Node.js, publicado como software libre y de código abierto bajo Licencia MIT. Está diseñado para la creación de aplicaciones web y API. Es el entorno de servidor estándar de facto para Node.js. Es relativamente mínimo con muchas funciones disponibles como complementos.

Proyecto Financiado por el Centro de Desarrollo Tecnológico Industrial (CDTI).
Expediente IDI-20150289
Cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Operativo Plurirregional de Crecimiento Inteligente 2014-2020

MongoDB

MongoDB es el software de bases de datos NoSQL con más aceptación y respaldo de los que están disponibles en la actualidad. Es un programa de base de datos orientado a documentos con formato (similar a) JSON, multiplataforma, libre y de código abierto.

MongoDB aplica un enfoque orientado a documentos para almacenar los datos. La idea es que todos los datos de una misma entidad pueden almacenarse como en forma de un único documento. Esto incluye la posibilidad de tener subdocumentos, los cuales en los sistemas de bases de datos relacionales (RDBMS) requieren de tablas separadas o ser almacenadas en forma de cadenas codificadas.

Los diferentes documentos se agrupan formando colecciones. El acceso a los documentos se obtiene mediante una clave única. Los documentos se almacenan mediante objetos en formato JSON o BSON binario

El objetivo de MongoDB es implementar un almacén de datos que proporcione un alto rendimiento, alta disponibilidad y escalado automático. MongoDB ofrece también un sitio web de almacenamiento de back-end para los sitios web con mucho tráfico, lo cual les permite almacenar comentarios de los usuarios, blogs, u otros artículos, ya que es rápido, escalable y fácil de instalar e implementar.

Algunas de las razones que motivan la gran aceptación recibida por MongoDB son:

- Debido a que MongoDB está orientado a documentos, los datos se almacenan en la base de datos en un formato afín tanto para el procesamiento desde el lado del servidor como del lado del cliente. Esto elimina la necesidad de convertir las filas de las tablas relacionales en objetos y viceversa.
- MongoDB es una de las bases de datos de más alto rendimiento disponibles.
- El modelo de replicación de MongoDB facilita mantener la escalabilidad, a la vez que ofrece un alto rendimiento.
- La estructura de MongoDB facilita el sharding horizontal mediante la compartición de los datos a través de múltiples servidores.
- MongoDB no acepta el uso de sentencias SQL de formularios web ya que los objetos se almacenan como tales y no mediante el uso de cadenas SQL.

Características principales

MongoDB soporta consultas por rango y por campo, búsquedas empleando expresiones regulares. Las consultas pueden devolver campos específicos de documentos, así como incluir funciones de JavaScript definidas por el usuario. Las consultas también se pueden configurar para devolver una muestra aleatoria de los resultados de un determinado tamaño.

Proyecto Financiado por el Centro de Desarrollo Tecnológico Industrial (CDTI).
Expediente IDI-20150289
Cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Operativo Plurirregional de Crecimiento Inteligente 2014-2020

Los campos de un documento MongoDB pueden ser indexados mediante índices primarios y secundarios.

MongoDB proporciona alta disponibilidad con conjuntos de réplicas. Un conjunto de réplicas se compone de dos o más copias de los datos. Cada miembro del conjunto de réplica puede actuar como réplica primaria o secundaria en cualquier momento. Todas las lecturas y escrituras se realizan en la réplica principal de manera predeterminada. Las réplicas secundarias mantienen una copia de los datos de la réplica primaria utilizando replicación incorporada. Cuando una réplica primaria falla, el conjunto de réplicas lleva a cabo de manera automática un proceso de elección para determinar cuál de las réplicas secundarias debe convertirse en primaria.

MongoDB facilita el escalado horizontal utilizando sharding ("fragmentación"). El usuario elige una clave shard, la cual determina cómo se distribuyen los datos en una colección. Los datos se dividen en rangos (en base a la clave shard) y se distribuyen a través de múltiples fragmentos, donde uno de ellos actúa como el maestro o principal y los demás como esclavos. Por otro lado, la clave shard se puede utilizar como hash para acceder a un fragmento.

MongoDB puede ejecutar a través de múltiples servidores, realizando balance de carga o duplicando los datos para mantener el sistema en funcionamiento en caso de fallo de hardware.

MongoDB puede ser utilizado como un sistema de ficheros con características de balanceo de carga y replicación de datos a través de varias computadoras para el almacenamiento de los ficheros. Esta función, denominada Sistema en Red de Ficheros (GridFS), se incluye con los controladores de MongoDB. MongoDB brinda funciones para la manipulación de ficheros y su contenido a los desarrolladores. GridFS divide el fichero en partes y guarda cada trozo como un documento independiente.

Se puede utilizar MapReduce para el procesamiento por lotes de los datos y para operaciones de agregación. El marco de agregación permite a los usuarios obtener resultados similares a los obtenidos mediante la cláusula GROUP BY de SQL. Los operadores de agregación pueden agruparse para formar una tubería similar a las de Unix. El entorno de agregación incluye el operador $lookup, el cual permite unir documentos que provienen de varios documentos, de forma similar a los operadores estadísticos tales como la desviación estándar.

Se puede utilizar JavaScript en las consultas, en las funciones de agregación (como MapReduce), y enviarlo directamente a la base de datos para ser ejecutado.

MongoDB soporta colecciones de tamaño fijo. Este tipo de colección mantiene el orden de inserción y, una vez que se alcanza el tamaño especificado, se comporta como una cola circular.

Proyecto Financiado por el Centro de Desarrollo Tecnológico Industrial (CDTI).
Expediente IDI-20150289
Cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Operativo Plurirregional de Crecimiento Inteligente 2014-2020

Librerías auxiliares y su funcionalidad

- async.js: es un módulo que brinda potentes funciones para trabajar con JavaScript de forma asíncrona. Se diseñó para ser usado con Node.js y ser instalado a través de la NPM (Node Package Manager).
- body-parser.js: extrae toda la parte del cuerpo de una solicitud entrante y la expone como parte del req.body para facilitar su uso
- connect-mongo.js: almacena la conexión con MongoDB durante una sesión de trabajo con Express and Connect
- chakram.js: es un entorno de prueba para API REST ofreciendo un estilo de prueba BDD (Behaviour Driven Development). Chakram permite escribir pruebas claras y completas, asegurando que los puntos finales REST JSON funcionen correctamente a medida que se va desarrollando y en el futuro
- cors.js: es un paquete de Node.js para proporcionar un middleware para Connect/Express que se puede utilizar para permitir Cross Origin Resource Sharing (CORS) con varias opciones.
- errorhandler.js: es un manejador de error middleware solo para desarrollo. Este middleware se diseñó para ser utilizado en un entorno de desarrollo, ya que todas las trazas de la pila de errores y detalles internos completos de cualquier objeto que haya pasado por este módulo serán enviados al cliente cuando ocurra un error. Cuando se proporciona un objeto a Express como un error, este módulo mostrará tanto como sea posible acerca de este objeto, y lo hará mediante el uso de negociación de contenido para la respuesta entre HTML, JSON y texto plano sin formato.
- http.js: es una biblioteca de utilidades HTTP para aplicaciones del lado del cliente.
- lodash.js: es una moderna biblioteca de utilidades de JavaScript que brinda modularidad, rendimiento, y extras.
- merge.js: permite combinar varios objetos en uno solo, con la posibilidad de crear un nuevo objeto clonado a partir de los anteriores. Es similar que jQuery.extend pero más flexible. Funciona con Node.js y el navegador.
- mocha.js: es un entorno de prueba JavaScript con muchas características que ejecuta sobre Node.js y en el navegador, facilitando las pruebas en modo asíncrono. Las pruebas se ejecutan en serie, lo que permite la presentación de informes flexible y precisos, mientras que asocia las excepciones no capturadas con los casos de prueba adecuados.
- mongoose.js: es una herramienta de modelado de objetos para MongoDB diseñado para funcionar en un entorno asíncrono. Proporciona una solución sencilla, basada en esquemas para modelar los datos de la aplicación. Incluye funciones para conversión de tipos, validación, generación de consultas, conexiones de lógica de negocios y más.
- morgan.js: se utiliza para registrar (i.e. log) detalles de la solicitud. Básicamente es un registrador que, ante cualquier solicitud que se realice, genera logs de forma automática.

Proyecto Financiado por el Centro de Desarrollo Tecnológico Industrial (CDTI).
Expediente IDI-20150289
Cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Operativo Plurirregional de Crecimiento Inteligente 2014-2020

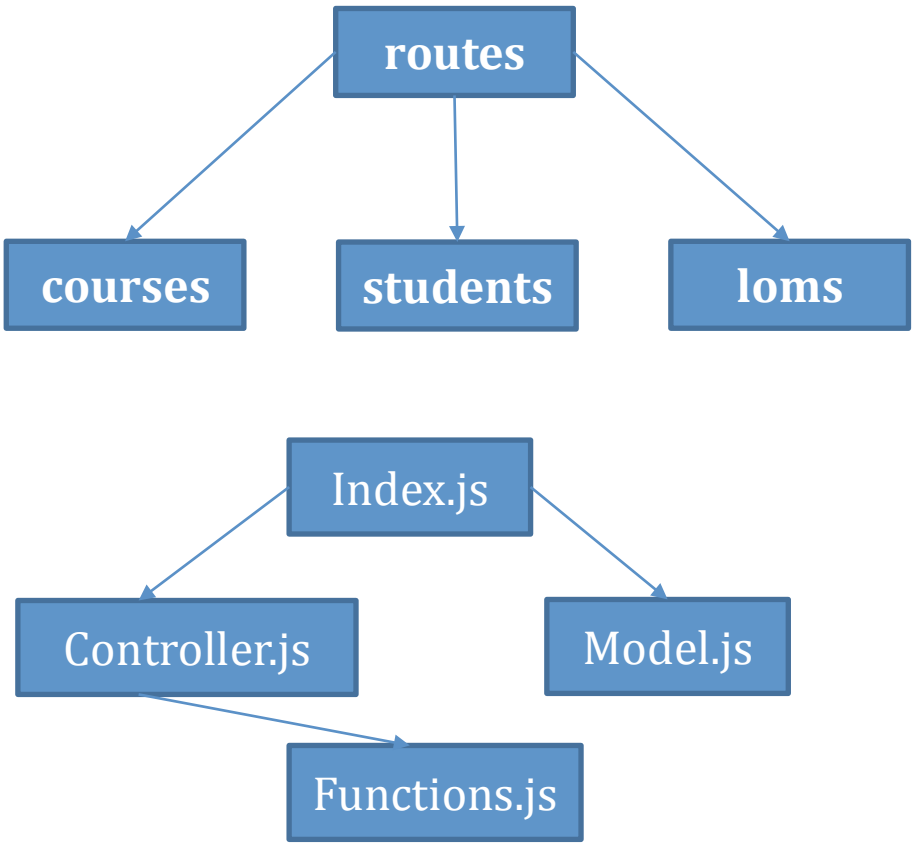
- multer.js: es un middleware de Node.js para el manejo de datos con multiples partes, la cual se utiliza sobre todo para la carga de archivos. Está escrito utilizando busboy (un analizador sintáctico para datos en formato HTML) para obtener una máxima eficiencia.
- sinon.js: sirve para espíar pruebas independientes, trozos e imitaciones de JavaScript. Sin dependencias, funciona con cualquier entorno de pruebas unitarias.
- path.js: es una biblioteca ligera de enrutamiento del lado del cliente que le permite crear aplicaciones de "una sola página" utilizando Hashbangs y/o HTML5 pushState. El módulo proporciona utilidades para trabajar con rutas de archivos y directorios.
- request.js: está diseñado para hacer llamadas HTTP de forma sencilla. Es compatible con HTTPS y sigue las redirecciones de forma predeterminada.
- supertest.js: biblioteca basada en SuperAgent para probar servidores de pruebas HTTP. Proporciona un alto nivel de abstracción para pruebas HTTP, permitiendo al mismo tiempo evitar la API de bajo nivel proporcionada por SuperAgent.
- supervisor.js: Un pequeño script supervisor para Node.js. Ejecuta el programa indicado permitiendo cargas de código en caliente (en ejecución), sin preocuparse por pérdidas de memoria y asegurando limpiar todas las referencias entre módulos, sin requerir un sistema completamente nuevo.
- yargs.js: ayuda a construir herramientas interactivas para comandos mediante el análisis de los argumentos y la generación de una interfaz de usuario elegante. Es una biblioteca Node.js

Proyecto Financiado por el Centro de Desarrollo Tecnológico Industrial (CDTI).
Expediente IDI-20150289
Cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Operativo Plurirregional de Crecimiento Inteligente 2014-2020

Estructura de los principales componentes

Los principales componentes del sistema están organizados de la forma siguiente:

1. **routes.js:** su objetivo es centralizar y uniformar la ruta de acceso al resto de los diferentes módulos de la aplicación.
2. **index.js**: maneja el inicio de la aplicación como un servidor web HTTP básico, el enrutamiento a los demás componentes y otras funciones de la aplicación. Establece la conexión entre la página web y las diferentes funciones de cada módulo de la aplicación.
3. **controller.js**: implementa los procedimientos que reciben los argumentos de las solicitudes enviadas a través de la página web y dan respuesta a dichas solicitudes.
4. **functions.js**: contiene las funciones auxiliares utilizadas de forma interna por los procedimientos del controller.js
5. **model.js**: describe el modelo de la base de datos empleada en cada módulo



Proyecto Financiado por el Centro de Desarrollo Tecnológico Industrial (CDTI).
Expediente IDI-20150289
Cofinanciado por el Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Operativo Plurirregional de Crecimiento Inteligente 2014-2020