

Pika-App

Complemento de Base de Datos

Grupo 12

Ángel Delgado Luna
Ezequiel Portillo Jurado

Contenido

Control de Versiones	3
Introducción	4
Requisitos	4
Requisitos de Información	4
Reglas de Negocio	5
Requisitos Funcionales	5
Requisitos No Funcionales	6
Relación con Datawarehouse	6
Investigación	6
Implantación sobre nuestra API y prueba realizada con Pentaho	8
Implementación	9
Metodología	9
Peticiones	13
Metodología y Organización del trabajo	15
Gestión de código	17
Stack tecnológico	17
Documentos	23
Integrantes del equipo	23
Bibliografía	24

Control de Versiones

Fecha	Motivo	Autor
8/05/2020	Stack Tecnológico Metodología Operativa con Pentaho Integrantes del equipo Implementación	Ezequiel Portillo Jurado
14/05/20	Introducción Requisitos	Ángel Delgado Luna
14/05/20	Actualización de la Implementación	Ezequiel Portillo Jurado
	Relación con Datawarehouse*	Ángel Delgado Luna

*La relación con Datawarehouse se ha realizado atendiendo a la bibliografía citada en el proyecto. Se han realizado resúmenes de los documentos y obtenido conclusiones que reflejamos en su apartado.

Introducción

Pika-App surge de la idea de poner en contacto a clientes con vehículos eléctricos con locales u entidades que dispongan de enchufes cualificados para poder recargarlos cuando lo necesiten.

Entre los vehículos abarcables por este proyecto están patinetes eléctricos, coches o bicicletas eléctricas. Otros elementos de la misma naturaleza como recarga de móviles (sin atender a la naturaleza propia del objetivo del proyecto) podría entrar en el contexto de este proyecto, no obstante nuestro foco se basa en el área de los vehículos

Para realizar una primera versión de este proyecto y dadas las tecnologías a emplear, se ha optado por la realización de una API Rest modulada con el framework Mongoose. Este, como se verá en su correspondiente apartado nos permitirá trabajar con los datos de una forma ordenada y estructurada para la base de datos sobre la que actuaremos. Principalmente, nos hemos interesado en implementar aspectos de registro de usuarios y valoración de locales. Demostrando por tanto un dominio de los CRUD con MongoDB y Mongoose así como de las operaciones (Queries en futuras menciones) que podemos hacer con Pentaho para trabajar los datos que albergamos en MongoDB.

Requisitos

Requisitos de Información

Caso de uso 1 – Gestión de usuarios

1. Las personas se registrarán en la plataforma con nombre, apellidos, usuario, contraseña y localidad.
2. Podrán registrarse dos tipos de usuarios: aquellos que tengan un local para ofrecer el servicio propuesto (1) y aquellos que posean vehículos que quieran beneficiarse (2).
3. Los usuarios con local (1) deberán incluir la identificación de este. Por el contrario, los usuarios propietarios de vehículos (2) deberán incluir una tarjeta de crédito para realizar los pagos.

Caso de uso 2 – Gestión de locales

1. Cada usuario con local, tendrá que registrar la dirección, capacidad de vehículos y su disponibilidad.

Caso de uso 3 – Gestión de vehículos

1. Cada vehículo tendrá un propietario, una carga máxima que servirá como información al propietario del local para conocer la capacidad del mismo y una matrícula.

Caso de uso 4 – Gestión de valoraciones

1. Los usuarios podrán valorar de 1 a 10 los locales por los que han sido atendidos.

Reglas de Negocio

Caso de uso 1 – Gestión de usuarios

1. El usuario de cada persona deberá ser un campo de identificación única.

Caso de uso 2 – Gestión de locales

Caso de uso 3 – Gestión de vehículos

1. La matrícula de cada vehículo deberá ser única además de ser un parámetro obligatorio.

Caso de uso 4 – Gestión de valoraciones

Requisitos Funcionales

Caso de uso 1 – Gestión de usuarios

1. Los usuarios podrán registrarse en la plataforma.
2. Cada usuario podrá obtener sus datos
3. Cada usuario podrá actualizar sus datos básicos (Entiéndase por datos básicos de aquellos no diferenciados por el tipo de usuario).

Caso de uso 2 – Gestión de locales

1. Los usuarios podrán crear locales
2. Los usuarios podrán visualizar todos los locales de la plataforma
3. Los usuarios podrán consultar locales concretos
4. Los usuarios podrán actualizar los datos de sus locales
5. Los usuarios podrán eliminar los locales

Caso de uso 3 – Gestión de vehículos

1. Los usuarios podrán crear vehículos
2. Los usuarios podrán visualizar todos los vehículos de la plataforma
3. Los usuarios podrán consultar vehículos concretos
4. Los usuarios podrán actualizar los datos de sus vehículos

Los usuarios podrán eliminar los vehículos

Caso de uso 4 – Gestión de valoraciones

1. Los usuarios crearan valoraciones sobre los locales.
2. Los usuarios podrán visualizar todas las valoraciones realizadas sobre los locales.

Requisitos No Funcionales

Caso de uso 1 – Gestión de usuarios

1. Para la gestión de usuarios y control de su acceso, se desea implementar la tecnología Json Web Token como modo de autenticación en el sistema para el uso de los diferentes recursos de la API a implementar.

Relación con Datawarehouse

Investigación

La concepción de uso de Datawarehouse se refiere al análisis, mediante inteligencia empresarial (Business Intelligent – BI, de ahora en adelante), de diferentes datos estructurados a través de un modelo relacional. Este aspecto es clave para comprender porque modelos NoSQL pueden suponer o no una controversia a la hora de usarlos para análisis de datos.

A lo largo de este apartado, nos centraremos en MongoDB por almacenar los datos con un formato de documento similar a las estructuras JSON. La complejidad de dicha estructura reside en la no posibilidad de uso del lenguaje SQL para sus consultas y análisis (lo que supondría una conversión JSON a SQL), así como en su posible no uniformidad a la hora de guardar los datos. Los clásicos datawarehouse no contemplan la posibilidad de uso para bases de datos semi-estructuradas y no estructuradas, aunque en su aplicación pudieran realizarse determinados arreglos para su funcionamiento.

Dejando a un lado el inconveniente de tener que traducir de JSON a SQL, la mayoría de las bases de datos NoSQL se basan en el concepto de pares clave-valor, que se fragmentan en nodos basados en una clave única.

En un sistema como MongoDB, hacer un escaneo completo de la tabla (que se requiere para muchas consultas analíticas) es una operación muy costosa. La agregación de datos, que también es comúnmente requerida para BI y análisis, requiere técnicas como “map” o “reduce” que pueden reunir datos de múltiples nodos dispares.

Muchos expertos están de acuerdo en que, aunque MongoDB puede soportar BI, no es la primera opción para su uso como almacén de datos de apoyo a las operaciones analíticas y de información. MongoDB es ante todo una base de datos OLTP (análisis de tendencias), y debe considerarse con cuidado para su uso en una solución para los fines de inteligencia artificial.

Dicho esto, MongoDB proporciona una solución para traducir JSON a SQL y permitir el acceso directo a través de las herramientas BI: MongoDB BI Connector (Es parte de la oferta avanzada de pago de MongoDB Enterprise). Este conector proporciona a la plataforma información sobre el esquema de la colección MongoDB relevante, recibe consultas SQL de la plataforma BI y las traduce a las consultas MongoDB apropiadas. A continuación, devuelve los resultados en un formato SQL que la plataforma BI puede consumir. Algunos ejemplos que actualmente se pueden utilizar son: Panoply, Stitch, SlamData, FiveTran, Knowi, Pentaho o Jasper.

Para todo lo anterior, datawarehouse basa su análisis en dos principales grupos. El primero sobre datos que son medibles (Costes, Temperaturas, etc) y el segundo los datos descriptivos o llamados en la jerga de datawarehouse dimensiones que atribuyen significado a los hechos a través de lo que entenderíamos como claves extranjeras.

Ante esto, una reestructuración completa de la base de datos no sería algo lógico por conllevar una dificultad técnica variable en función del esquema de datos a trabajar. No obstante, un proceso de denormalización basado en determinados puntos podría ayudar a lidiar con dicha tarea:

- Estructuras de 1:N (OneToMany) son triviales de tratar, por su relación directa con los elementos.
- Para aquellas relaciones que supongan una determinada complejidad, como las auto-referenciables o también llamadas relaciones involutivas, se recomienda seguir un patrón como el anterior.
- Relaciones M:N (ManyToMany) corren el riesgo de perder información en el proceso de denormalización o reestructuración parcial de datos dado el volumen de datos que tienen que trabajar y la posible redundancia de los mismos.
- Cuantos menos niveles, de carácter jerárquico, tenga la estructura, más fácil de generar y trabajar los datos será. Esto permite un análisis más exacto y preciso que se reflejará a posteriori en los informes que se generen.

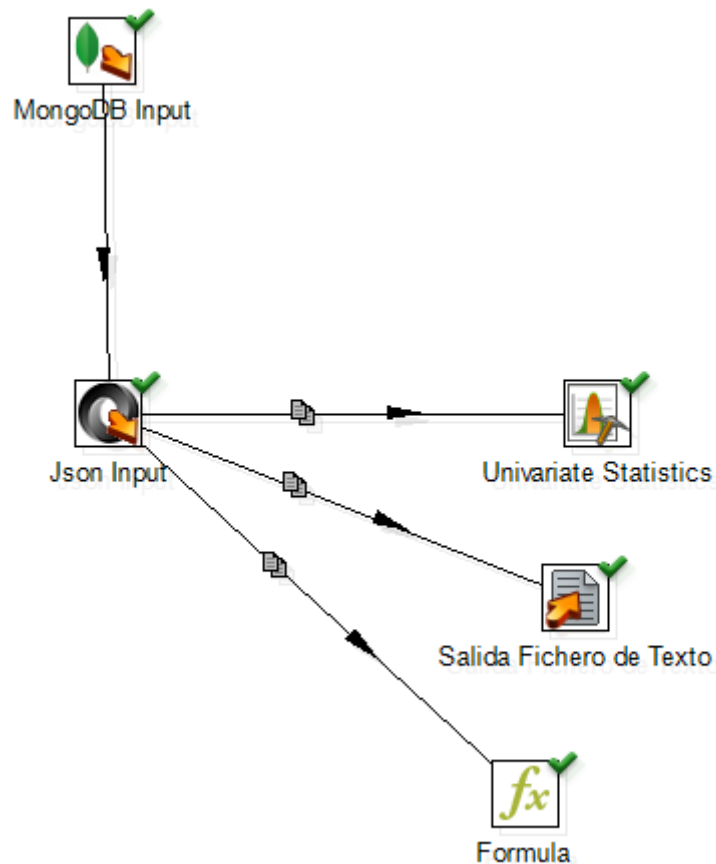
Todo esto, será posible atendiendo al principio de uniformidad citado. Es decir, al permitir las bases de datos como MongoDB coexistir en la misma colección, documentos con diferentes campos, puede incurrir en un análisis impreciso. Por esto, operar con los datos matemáticamente (por ejemplo: con operaciones de media o desviación estándar) se hace una ardua tarea.

Con esto concluimos que MongoDB en el aspecto de datawarehouse, sería una apuesta segura con muchas restricciones (entre ellas, las citadas anteriormente). Por ello, en virtud de este trabajo, se emplea el framework “Mongoose” como ODM gestor de los datos. Esto, nos permite mantener una uniformidad con los datos que estamos trabajando y así hacer el proceso de denormalización lo más seguro y completo posible para obtener los mejores resultados. Un ejemplo, además del proyecto que planteamos de nuestra API Rest, sería el citado en la bibliografía [1].

A modo resumido, el artículo nos introduce al ámbito sanitario el uso de bases de datos NoSQL para trabajar con métricas elaboradas por diferentes marcapasos. A través de una API Rest se envían los datos que va midiendo cada uno de ellos y los almacena en la colección de datos. Dado el potencial de las bases de datos NoSQL ante un gran volumen de datos y el mantenimiento de la estructura continua, permiten que MongoDB (como sistema gestor) actúe como datawarehouse consiguiendo un análisis preciso en un tiempo menor que cuando usaban un esquema relacional en sus primeros análisis.

Implantación sobre nuestra API y prueba realizada con Pentaho

Como hemos indicado antes, es posible utilizar MongoDB con datawarehouse para el análisis de datos. En nuestro caso, a través de nuestra API, vamos a realizar un pequeño análisis de las valoraciones realizadas por los clientes a los locales almacenadas en nuestra base de datos.



Paso 1. Entrada de Datos de MongoDB

Step name: MongoDB Input

Configure connection | Input options | Query | Fields

Database: pika-database [Get DBs]

Collection: valoraciones [Get collections]

Read preference: primary

Paso 2. Entradas JSON a analizar


Rows of step: Json Input (6 rows)

#	json
1	[{"_id": {"\$oid": "Sebd2ce932cdf530ec119403"}, "valoracion": 8, "local": {"\$oid": "Sebd2bf332cdf530ec119400"}, "usuario": {"\$oid": "Sebd27b032cdf530ec1193eb"}, "createdAt": {"\$date": "2020-05-14T11:34:01.578Z"}, "updatedAt": {"\$date": "2020-05-14T11:34:01.578Z"}}, {"_id": {"\$oid": "Sebd2cc532cdf530ec119404"}, "valoracion": 7, "local": {"\$oid": "Sebd2c2932cdf530ec119402"}, "usuario": {"\$oid": "Sebd2c2932cdf530ec119402"}, "createdAt": {"\$date": "2020-05-14T11:34:29.167Z"}, "updatedAt": {"\$date": "2020-05-14T11:34:29.167Z"}}, {"_id": {"\$oid": "Sebd2cf932cdf530ec119405"}, "valoracion": 9, "local": {"\$oid": "Sebd2c2932cdf530ec119402"}, "usuario": {"\$oid": "Sebd28ed32cdf530ec1193ef"}, "createdAt": {"\$date": "2020-05-14T11:34:49.873Z"}, "updatedAt": {"\$date": "2020-05-14T11:34:49.873Z"}}, {"_id": {"\$oid": "Sebd2cf032cdf530ec119406"}, "valoracion": 3, "local": {"\$oid": "Sebd2bf332cdf530ec119400"}, "usuario": {"\$oid": "Sebd27b032cdf530ec1193eb"}, "createdAt": {"\$date": "2020-05-14T11:35:12.973Z"}, "updatedAt": {"\$date": "2020-05-14T11:35:12.973Z"}}, {"_id": {"\$oid": "Sebd2d0432cdf530ec119407"}, "valoracion": 3, "local": {"\$oid": "Sebd2c0e32cdf530ec119401"}, "usuario": {"\$oid": "Sebd28bc32cdf530ec1193ef"}, "createdAt": {"\$date": "2020-05-14T11:35:32.812Z"}, "updatedAt": {"\$date": "2020-05-14T11:35:32.812Z"}}, {"_id": {"\$oid": "Sebd2d1d32cdf530ec119408"}, "valoracion": 6, "local": {"\$oid": "Sebd2c0e32cdf530ec119401"}, "usuario": {"\$oid": "Sebd28ed32cdf530ec1193ef"}, "createdAt": {"\$date": "2020-05-14T11:35:57.837Z"}, "updatedAt": {"\$date": "2020-05-14T11:35:57.837Z"}}]

Paso 3. Estadísticas

valoracion(stdDev)	valoracion(min)	valoracion(max)	valoracion(median)
2,5	3	9	6,5

Paso 4. Salida de fichero post-análisis

 datos.txt: Bloc de notas

```
Archivo  Edición  Formato  Ver  Ayuda
valoracion
8,0
7,0
9,0
3,0
3,0
6,0
```

Paso 5. Aplicación de fórmula para valorar la valoración

	valoracion	buena_mala_nota
	8,0	Nota buena
	7,0	Nota buena
	9,0	Nota buena
	3,0	Nota mala
	3,0	Nota mala
	6,0	Nota buena

Para considerar si una nota es buena o mala se ha marcado el límite en 5. Considerando que, si la nota es igual o superior es buena, en otro caso será mala.

Implementación

Metodología

Para el desarrollo del proyecto se ha decidido seguir la siguiente estructura de carpetas para organizar las distintas funcionalidades de esta.

```
✓ pika-app
  ✓ app
    > controllers
    > models
    > routes
```

En primer lugar, encontramos los controladores. Cada uno de los esquemas creados en base de datos tiene un controlador, que es donde se implementan los métodos que quieras para ese esquema, típicamente aquí es donde nos encontramos el CRUD.

```
exports.create = (req, res) => {

  //Comprobar respuesta valida
  if(!req.body.matricula){
    return res.status(400).send({
      message: "La matrícula del vehículo no puede estar vacío"
    });
  }

  //Crear un vehiculo
  const vehiculo = new Vehiculo({
    propietario: ObjectId(req.body.propietario),
    tipo: req.body.tipo,
    cargaMaxima: req.body.cargaMaxima || "Sin carga máxima",
    matricula: req.body.matricula
  });

  //Guardar en base de datos el vehiculo
  vehiculo.save().then(data => {
    res.send(data);
  }).catch(err => {
    res.status(500).send({
      message: err.message || "Algún error ha ocurrido creando el vehiculo."
    });
  });
};
```

Aquí se ve como está implementado el método de crear vehículo. Como se ve en el código aquí se añaden todas las restricciones que necesites y puedes mandar distintos tipos de errores según

no se vaya pasando cada una de las validaciones. Si todo es correcto se crea el esquema de vehículo con los datos y se guarda en base de datos.

En segundo lugar, se encuentra la carpeta de modelos. Aquí es donde se declara la forma que tendrá el esquema además de algunas posibles restricciones.

```
const mongoose = require('mongoose');
var uniqueValidator = require('mongoose-unique-validator');
var TipoVehiculo = require('../models/tipoVehiculo.model.js');
mongoose.model('TipoVehiculo');

require('../tipoVehiculo.model');
require('../routes/tipoVehiculo.routes');

//Crea el esquema de la clase
const VehiculoSchema = mongoose.Schema({
  propietario: {
    type: mongoose.Schema.Types.ObjectId,
    refPath: 'User.__t'
  },
  tipo: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'TipoVehiculo'
  },
  cargaMaxima: String,
  matricula: {
    type: String, index: true, unique: true, required: true
  }
});
//Añade automáticamente createdAt u UpdatedAt
}, {
  timestamps: true,
  autoCreate: true
});

VehiculoSchema.plugin(uniqueValidator);

module.exports = mongoose.model('Vehiculo', VehiculoSchema);
```

Aquí se ve el ejemplo del esquema creado para la clase vehículo. Además de declarar el nombre de los atributos que contendrá este esquema también podemos añadir otras restricciones como el ser único, si puede ser vacío...

Finalmente se exporta el modelo con un nombre para que pueda ser llamado en otros archivos.

En tercer y último lugar encontramos la carpeta de rutas. En este archivo se declara la ruta que tendrá cada uno de los métodos para que pueda ser llamado.

```
module.exports = (app,middleware) => {
  const vehiculos = require('../controllers/vehiculo.controller.js');

  // Create a new vehiculo
```

```

app.post('/vehiculos', middleware, vehiculos.create);

// Retrieve all vehiculos
app.get('/vehiculos', middleware, vehiculos.findAll);

// Retrieve a single vehiculo with vehiculoId
app.get('/vehiculos/:vehiculoId', middleware, vehiculos.findOne);

// Update a vehiculo with vehiculoId
app.put('/vehiculos/:vehiculoId', middleware, vehiculos.update);

// Delete a vehiculo with vehiculoId
app.delete('/vehiculos/:vehiculoId', middleware, vehiculos.delete);
}

```

Aquí se ve un ejemplo de las rutas utilizadas para el CRUD completo del esquema vehículo.

Estas son los principales archivos que componen el proyecto además de varios archivos de configuración de Node. Cabe destacar también la existencia del archivo server.js en el cual se declaran las rutas principales para llamar a los métodos.

JS server.js

Como se ha podido observar en los métodos de la ilustración anterior, las rutas de dicha API están securizadas a través de un “middleware”. Este actúa como mecanismo de seguridad para garantizar que las peticiones se realizan con un token válido. La técnica empleada ha sido Json Web Token, la cual ha consistido en la implementación de determinados ficheros:

- config/jwt.config.js

Se ha definido una clave llamada “llave” para la generación del token.

- middleware.js

Actúa como intermediario y se encarga de revisar que el token utilizado en la petición HTTP a través de la cabecera “access-token” es válido.

- authentication.route.js

Ruta de acceso para solicitar autenticación en el sistema

- authentication.controller.js

Generador de token en caso de que el JSON de autenticación remitido por petición POST sea válido. Este únicamente lo considera válido si el par username-password devuelve un objeto no nulo.

- server.js (Como adaptación para cada una de las rutas)

Paso del middleware a cada una de las rutas. Diferenciando entre aquellas que son de carácter público (la de autenticación) y el resto de carácter privado.

Para la implementación de JWT se ha consultado tanto proyectos anteriores basados en NodeJS que uno de los integrantes había realizado como en otros recursos. [2]

En cuanto a los mecanismos de herencia para los objetos usuarios, hemos seguido un esquema de generalización mediante discriminadores (así llamado por Mongoose).

Paso 1. Creamos una función que tenga los parámetros básicos.

```
function BaseSchema(add){
  var UserSchema = mongoose.Schema({
    userAccount: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'UserAccount'
    },
    name: String,
    surnames: String,
    location: String
  }, {
    timestamps: true,
    autoCreate: true,
    strict: false
  });

  if(add){
    UserSchema.add(add);
  }

  return UserSchema;
}
```

Paso 2. Marcamos el esquema como herencia a través del paquete Util

```
util.inherits(BaseSchema, Schema);
```

Paso 3. En base al esquema/función definida anterior, definimos el criterio de discriminación junto a los atributos que compondrán cada uno de los objetos heredados.

```
var UserSchema = new BaseSchema();

UserSchema.virtual('type').get(function () { return this.__t; });

var User = mongoose.model('User', UserSchema);
module.exports = User;

var usuarioVehiculo = new BaseSchema({creditCard:{type:String}});
module.exports = User.discriminator('UsuarioVehiculo', usuarioVehiculo);
```

```
var usuarioLocal = new BaseSchema({localNumber:{type:String}});
module.exports = User.discriminator('UsuarioLocal', usuarioLocal);
```

Peticiones

Petición	JSON	Retorno
Autenticación	<pre>{ "username": "", "password": "" }</pre> <p>POST http://localhost:3000/authenticate</p>	Token generado
Crear vehículo	<pre>{ propietario: <Id Usuario>, tipo: <Id Tipo>, cargaMaxima: " ", matricula: " " }</pre> <p>POST http://localhost:3000/vehiculos</p>	Vehículo creado
Obtener un vehículo	GET http://localhost:3000/vehiculos/:vehiculoid	Vehículo solicitado
Editar vehículo	<pre>{ propietario: <Id Usuario>, tipo: <Id Tipo>, cargaMaxima: " ", matricula: " " }</pre> <p>PUT http://localhost:3000/vehiculos/:vehiculoid</p>	Vehículo editado
Eliminar vehículo	DELETE http://localhost:3000/vehiculos/:vehiculoid	Vehículo eliminado
Listar vehículos	GET http://localhost:3000/vehiculos/	Lista de vehículos
Crear local	<pre>{ propietario: <Id propietario>, direccion: " ", numMaxVehiculo: 120, disponible: true-false }</pre> <p>POST http://localhost:3000/local</p>	Local creado

Obtener un vehículo	GET http://localhost:3000/:localId	Locales obtenido
Editar local	<pre>{ propietario: <Id propietario>, direccion: " ", numMaxVehiculo: 120, disponible: true-false }</pre> PUT http://localhost:3000/local	Local editado
Listar locales	GET http://localhost:3000/local	Locales creados
Eliminar locales	DELETE http://localhost:3000/:localId	Local eliminado
Crear valoración	<pre>{ valoracion: [1-10], local: <Id Local>, usuario: <Id Usuario> }</pre> POST http://localhost:3000/valoracion	Valoración creada
Listar valoración	GET http://localhost:3000/valoracion	Obtener todas las valoraciones realizadas
Crear usuario	POST http://localhost:3000/user	Usuario Creado
	UsuarioLocal <pre>{ username: "", password:"", name: "nombre", surnames: "apellidos", location: "localidad", localNumber:"numeroLocal" }</pre>	
	UsuarioVehiculo <pre>{ username: "", password:"", name: "nombre", surnames: "apellidos", location: "localidad", creditCard:"ES123456789" }</pre>	
Obtener un usuario	GET http://localhost:3000/local/:userId GET http://localhost:3000/vehiculo/:userId	Usuario solicitado
Editar usuario	<pre>{ userAccount: <Id Cuenta Usuario>, name: "nombre", surnames: "apellidos", location: "localidad" }</pre>	Usuario editado

	<pre> } PUT http://localhost:3000/local/:userId PUT http://localhost:3000/vehiculo/:userId </pre>	
Crear tipo de vehiculo	<pre> { "nombreTipo": "Mi tipo de vehiculo" } POST http://localhost:3000/tipoVehiculo </pre>	TipoVehiculo creado
Editar tipo de vehiculo	<pre> { "nombreTipo": "RUEDA" } PUT http://localhost:3000/tipoVehiculo/:tipoid </pre>	Tipo Vehiculo editado
Listar tipo de vehículos	GET http://localhost:3000/tipoVehiculo	Listado de tipo de vehículos
Eliminar tipo de vehiculo	DELETE http://localhost:3000/tipoVehiculo/:tipoid	Eliminado tipo de vehiculo

Metodología y Organización del trabajo

Durante la realización del proyecto nos hemos organizado siguiendo en orden la siguiente lista de tareas:

1. Investigación de las nuevas tecnologías y montaje del entorno de desarrollo

Los dos integrantes del grupo han buscado documentación relacionada con MongoDB y con Mongoose, siempre intentando buscar la forma de implementar una API, ya que era nuestro principal objetivo.

Estos documentos han sido almacenados y analizados durante una reunión, además de revisar las clases de teoría o práctica donde se hayan usado alguna de estas tecnologías.

Finalmente, cuando se han tenido los conocimientos necesarios para empezar, se descargaron todas las herramientas y se montó un entorno de desarrollo común para evitar futuros problemas de implementación.

2. Creación de un modelo para nuestra API

Los integrantes del grupo realizaron una reunión para decidir modelo de base de datos que tendría la API basándose en la idea de recarga de vehículos eléctricos.

Cuando este modelo estuvo completo se dividió en partes equitativas para los dos integrantes del grupo.

3. Implementación de la API

Los integrantes del grupo implementaron todo el código referente a la API. Este código se iba subiendo poco a poco a un repositorio privado hasta que se finalizó.

4. Realización de la documentación aportando todos los datos aprendidos durante la implementación e investigación del proyecto

Se realizó una reunión para decidir los puntos que tendría la documentación del proyecto. Estos puntos se dividieron equitativamente entre los integrantes del grupo.

5. Trabajo con Pentaho para aplicación de Datawarehouse al esquema de bases de datos no relacional implementado.

Relación de MongoDB con Pentaho para el análisis de los datos propuestos

6. Conexión de datawarehouse con relación a MongoDB y Mongoose

Valorar pros y contras de como un esquema de inteligencia empresarial como datawarehouse compatibiliza con el sistema no relacional.

7. Realización de la presentación

Siguiendo la estructura planteada en la documentación se realizó una reunión donde entre los dos integrantes del grupo se formó dicha presentación.

8. Realización del video demostración

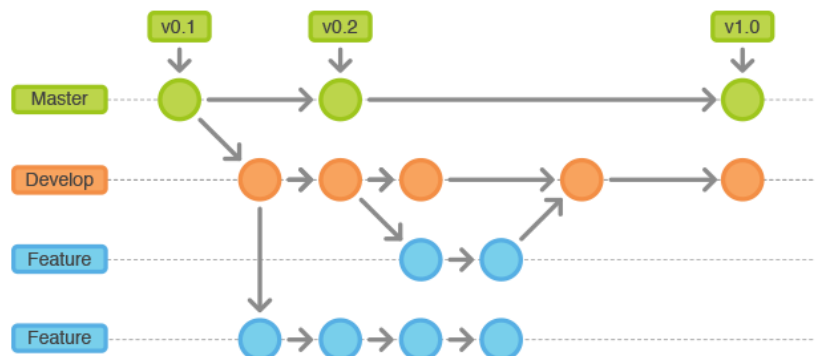
Realización del video de la API y su interacción con Pentaho.

Gestión de código

GitHub es un sistema de gestión de proyectos y control de versiones de código, así como una plataforma de red social diseñada para desarrolladores.

Esta plataforma se ha utilizado para subir el código generado durante la fase de desarrollo del proyecto para una mejor organización y para su fácil fusión entre las partes de cada integrante.

La gestión de código se ha realizado con una versión un poco modificada de GitFlow a través de un repositorio de github (<https://github.com/EzequielPJ/picaApp>)



El flujo de trabajo con GitFlow se basa principalmente en dos ramas, la rama *develop* y la rama *master*. La rama *develop* es aquella rama donde convergen las ramas de desarrollo y la rama *master* que es nuestra rama principal o de producción, a partir de la cual van a crearse

algunas ramas como, por ejemplo, *hotfix/importError*, que se crean para solucionar errores lo antes posible, o como *feature/vehículo*, rama donde se implementaría el trabajo relacionado con el esquema vehículo.

En nuestro caso se ha seguido esta gestión descrita quitando el caso de la rama develop, ya que al ser solo 2 integrantes los que componen el grupo no vimos necesario crear esta rama. Por tanto, todas las ramas feature y fix salen y entran directamente en master.

Stack tecnológico

MongoDB

MongoDB es un sistema de base de datos NoSQL orientado a documentos de código abierto y escrito en C++, que en lugar de guardar los datos en tablas lo hace en estructuras de datos BSON (similar a JSON) con un esquema dinámico. Al ser un proyecto de código abierto, sus binarios están disponibles para muchísimos sistemas operativos diferentes y es usado en múltiples proyectos o implementaciones en empresas.

El elemento principal de MongoDB es como almacena la información. MongoDB almacena toda la información en documentos JSON.

```
{  propietario: "ezequiel"
  matricula: "2456UY"
  tipoVehiculo: "Coche eléctrico" }
```



El almacenar la información en documentos JSON permite a MongoDB tener independencia del schema de almacenamiento, es decir, pueden existir más o menos campos en el documento dentro de una misma colección de documentos. Una de las cosas importantes de los documentos es que estos van tipados. Además, los documentos nos permiten nuevas estructuras como arrays o subdocumentos que permitirán que de una sola consulta se recupere toda la información y evite así la necesidad de ejecutar consultas de tipo join.

Las principales características de MongoDB son:

Alto rendimiento

El alto rendimiento para la persistencia en MongoDB se basa en dos puntos: La posibilidad de tener documentos con la información anidada, evitando, de esta forma, un número elevado de operaciones de I/O. Y el soporte de índices y la posibilidad de crear índices sobre arrays y subdocumentos.

Alta disponibilidad

MongoDB proporciona alta disponibilidad mediante la réplica automática conocida como *replica set*, la cual proporciona redundancia de datos y failover automático, es decir, la transferencia automática a un nuevo nodo cuando se encuentra un fallo en uno de los nodos.

Escalado Automático

MongoDB nos ofrece un escalado horizontal. Para ello el *sistema de sharding* nos permite distribuir información por diferentes cluster de máquinas.

Mongoose

Mongoose es una biblioteca de JavaScript que le permite definir esquemas con datos fuertemente tipados. Una vez que se define un esquema, Mongoose le permite crear un Modelo basado en un esquema específico.

Se trata de un Object Document Mapper (ODM). Esto significa que Mongoose le permite definir objetos con un esquema fuertemente tipado que se asigna a un documento MongoDB.

Este ODM proporciona una increíble cantidad de funcionalidades para crear y trabajar con esquemas. Mongoose actualmente contiene ocho SchemaTypes que una propiedad se guarda como cuando se conserva a MongoDB. Son:

1. String (Cadena)
2. Number (Número)
3. Date (Fecha)
4. Buffer
5. Boolean (Booleano)
6. Mixed (Mixto)
7. ObjectId
8. Array (Matriz)



Además de estas opciones comunes, ciertos tipos de datos le permiten personalizar aún más cómo se almacenan y recuperan los datos de la base de datos. Por ejemplo, un tipo de datos String también le permite especificar las siguientes opciones adicionales:

- convertirlo a minúsculas
- convertirlo a mayúsculas
- recortar datos antes de guardar
- una expresión regular que puede limitar los datos que se pueden guardar durante el proceso de validación
- una enumeración que puede definir una lista de cadenas que son válidas

Las propiedades Número Number y Fecha Date son compatibles con la especificación de un valor mínimo y máximo permitido para ese campo.

La mayoría de los ocho tipos de datos permitidos deberían serle familiares. Sin embargo, hay varias excepciones que pueden surgirle, como Buffer, Mixed y ObjectId.

El tipo de datos Buffer le permite guardar datos binarios. Un ejemplo común de datos binarios sería una imagen o un archivo codificado, como un documento PDF.

El tipo de datos Mixed convierte la propiedad en un campo "todo vale". Este campo se parece a cuántos desarrolladores pueden usar MongoDB porque no hay una estructura definida. Tenga cuidado con el uso de este tipo de datos ya que pierde muchas de las excelentes funciones que ofrece Mongoose, como la validación de datos y la detección de cambios de entidades para saber automáticamente si desea actualizar la propiedad al guardar.

El tipo de datos ObjectId comúnmente especifica un enlace a otro documento en su base de datos. Por ejemplo, si tiene una colección de libros y autores, el documento del libro puede contener una propiedad ObjectId que hace referencia al autor específico del documento.

Postman

Postman nace como una herramienta que principalmente nos permite crear peticiones sobre APIs de una forma muy sencilla y poder, de esta manera, probar las APIs. Todo basado en una extensión de Google Chrome. El usuario de Postman puede ser un desarrollador que esté comprobando el funcionamiento de una API para desarrollar sobre ella o un operador el cual esté realizando tareas de monitorización sobre un API.



Alrededor de la idea de testear las APIs, Postman nos ofrece un conjunto de utilidades adicionales para poder gestionar las APIs de una forma más sencilla. Es por ello que nos va a proporcionar herramientas para documentar los APIs, realizar una monitorización sobre las APIs, crear equipos sobre un API para que trabajen de forma colaborativa... convirtiendo a Postman plataforma de desarrollo de APIs que se basa por un modelo de desarrollo API First.

Siendo sus principales características las siguientes:

Características de Postman

- Crear Peticiones, te permite crear y enviar peticiones http a servicios REST mediante una interfaz gráfica. Estas peticiones pueden ser guardadas y reproducidas a posteriori.
- Definir Colecciones, mediante Postman podemos agrupar las APIs en colecciones. En estas colecciones podemos definir el modelo de autenticación de las APIs para que se añada en cada petición. De igual manera podemos ejecutar un conjunto de test, así como definir variables para la colección.
- Entorno Colaborativo, permite compartir las API para un equipo entre varias personas. Para ello se apoya en una herramienta de colaborativa en Cloud.

En resumen, esta herramienta ha facilitado la forma de probar nuestra API.

NodeJS

La aparición de Node.js ha supuesto toda una revolución en el mundo de JavaScript. Node.js es un entorno de código abierto pensado para hacer posible el desarrollo de aplicaciones JavaScript

del lado del servidor. Lo que permite que una vez desarrolladas puedan ejecutarse dentro en diferentes sistemas como OS X, Microsoft Windows y Linux.

Node.js también proporciona una nutrida biblioteca con módulos JavaScript que simplifica el desarrollo de aplicaciones web usando este framework en gran medida, el cual se ha utilizado para crear la API de vehículos eléctricos.

Las siguientes son algunas de las características importantes que hacen de Node.js la primera opción para los arquitectos de software.

- Asíncrono y controlado por eventos: todas las API de la biblioteca Node.js son asíncronas, es decir, sin bloqueo. Esencialmente significa que un servidor basado en Node.js nunca espera que una API devuelva datos. El servidor pasa a la siguiente API después de llamarlo y un mecanismo de notificación de Events of Node.js ayuda al servidor a obtener una respuesta de la llamada API anterior.
- Muy rápido: al estar construido en el motor JavaScript V8 de Google Chrome, la biblioteca Node.js es muy rápida en la ejecución de código.
- Procesos en un solo hilo, pero altamente escalable: Node.js utiliza un modelo de un solo hilo con bucle de eventos. El mecanismo de eventos ayuda al servidor a responder sin bloqueos y hace que el servidor sea altamente escalable en comparación con los servidores tradicionales que crean hilos limitados para manejar las solicitudes.



Pentaho

Pentaho BI Suite es una herramienta de Business Intelligence que extrae y analiza datos con técnicas ETL (extraer, transformar y cargar, en sus siglas en inglés). Estos datos se muestran a posteriori en Cuadros de Mandos, que serán muy útiles para crear informes y tener un seguimiento de la consecución de objetivos.

En el caso de Pentaho, estamos ante un software open source que cuenta con herramientas para Big Data y IoT.

Pentaho tiene muchas herramientas diferentes entre las cuales cabe destacar las siguientes: Pentaho Business Analytics (Pentaho BA), que sirve para generar informes o cuadro de mandos, CTools que se usa para crear Dashboards, o Pentaho Data Integration (PDI) que ofrece datos analíticos muy precisos, sin apenas opción a error en la codificación de los mismos.

Pentaho BI es una herramienta imprescindible para la toma de decisiones empresariales gracias a su poderoso motor de análisis. Con la suite de Pentaho BI podremos generar multitud de informes en cualquier formato: dinámicos, adhoc, guiados, predefinidos etc...

Estos informes pueden ser de gran ayuda, por ejemplo, a la hora presentar reportes mensuales de ventas, lo que sería un informe estático al uso. Un escalón más allá tendríamos los informes dinámicos que nos permitirán interactuar con dicha información y analizar dicha información en detalle.

Las principales características que te aporta pentaho son:

1. Mayor accesibilidad a la información corporativa: Gracias a Pentaho Data Integration (PDI ó Kettle) tendremos contenidos más accesibles, comprensibles y navegables. Lo que mejorará la productividad de la empresa.
2. Aporta una seguridad extra a la información de la empresa: Con Pentaho BI podremos controlar el acceso a la información mediante la asignación de roles y además, podremos saber quién hace uso de los datos y de qué datos exactamente.
3. Es la base para la toma de decisiones: La ordenación de toda la información y los datos recogidos, además de la correcta presentación de los mismos para su posterior análisis, nos permitirá, como ya hemos indicado, tomar las decisiones de manera más acertada.

En concreto para nuestra implementación y prueba de Pentaho hemos utilizado Spoon, una Interfaz Gráfica de Usuario (GUI), que permite diseñar transformaciones y trabajos que se pueden ejecutar con las herramientas de Kettle (Pan y Kitchen).

Gracias a Spoon vamos a poder realizar procesos de ETL de manera fácil y rápida. En concreto, nos va a permitir hacer Data Warehouse, con estructura en Estrella, pudiendo formar las tablas de Hechos y Dimensiones y sus relaciones entre ellas sin ninguna dificultad.

Prácticamente nos va a permitir hacer de todo: crear conexiones a los datos, hacer todo tipo de transformaciones, insertar fórmulas, transformaciones directas ya implementadas gracias a su calculadora... Sencillamente en pocos pasos te permite hacer muchas cosas sin tener que programar directamente con código.



Documentos

En cuanto a las herramientas utilizadas para gestionar documentos del proyecto se han utilizado Google Drive y Microsoft Office. Drive es un servicio de almacenamiento de archivos el cual se ha utilizado para compartir entre los miembros del grupo toda la documentación referente a los proyectos como esquemas UML, partes de documentación, decisiones de reuniones... Su principal funcionalidad ha sido compartir archivos entre los integrantes del grupo.

Integrantes del equipo



Ángel Delgado Luna

4º Ingeniería del Software

“Me ha gustado este trabajo ya que pude aprender nuevas tecnologías”

Implementación, Documentación y Presentación

Ezequiel Portillo Jurado

4º Ingeniería del Software

“Me ha gustado este trabajo porque hemos tratado tecnologías que se usan actualmente en el mercado”

Implementación, Documentación y Presentación



Bibliografía

- [1] MongoDB as a Data Warehouse: Time Series and Device History Data (Medtronic) Transcript. <https://www.mongodb.com/mongodb-data-warehouse-time-series-and-device-history-data-medtronic-transcript>
- [2] Autenticando un API Rest con NodeJS y JWT. <https://medium.com/@asfo/autenticando-un-api-rest-con-nodejs-y-jwt-json-web-tokens-5f3674aba50e>
- [3] MongoDB y Conectores Bussiness Inteligent. <https://blog.panoply.io/mongodb-and-bi-friends-or-foes>
- [4] Artículo de Ciencia sobre el uso de datawarehouse en esquemas NoSQL <https://www.sciencedirect.com/science/article/pii/S1877050917300819>
- [5] Forum acerca de considerar MongoDB como buena solución de datawarehouse para inteligencia de negocio. <https://www.quora.com/Can-MongoDB-be-a-good-solution-for-a-data-warehouse-for-BI-If-yes-which-BI-visualisation-vendors-support-MongoDB-correctly>
- [6] ¿Qué es MongoDB?. <http://www.manualweb.net/mongodb/que-es-mongodb/>
- [7] Introducción a Mongoose. <https://code.tutsplus.com/es/articles/an-introduction-to-mongoose-for-mongodb-and-nodejs--cms-29527>
- [8] ¿Qué es Postman?. <http://www.arquitectoit.com/postman/que-es-postman/>
- [9] ¿Qué es NodeJs? <https://ifgeekthen.everis.com/es/que-es-node-js-y-primeros-pasos>
- [10] ¿Qué es Pentaho? <https://www.itop.academy/blog/item/que-es-pentaho-data-integration-pdi-y-para-que-sirve.html>
- [11] ¿Qué es Pentaho? Sus productos y ventajas. <https://www.incentro.com/es-es/blog/stories/que-es-pentaho/>