# Tema 2. Acceso a datos con Node.js

*Desarrollo de una API REST con Express y MongoDB*

# Ejemplo de aplicación 1

- Ejemplo con Express y Mongoose para proporcionar servicios básicos sobre una base de datos de contactos: *ProyectosNode/PruebaContactosExpressMongoDB/index.js*

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
```

  - **Librería BodyParser (de Node.js)**, traduce el cuerpo de las peticiones (JSON, query-string) antes de usarlas (version 4.16 o superiores express lo incluye por defecto)

# Ejemplo de aplicación 1

- Conexión con la BD

```
mongoose.connect(
    'mongodb://localhost:27017/contactos',
    {useNewUrlParser: true, useUnifiedTopology: true }
);
```

## Ejemplo de aplicación 1

- Definición del esquema de nuestra colección
  - Los esquemas son la estructura de una colección
  - A continuación, debemos crear un modelo a partir de este esquema.

```
let contactoSchema = new mongoose.Schema({
    nombre: {
        type: String,
        required: true,
        minlength: 1,
        trim: true
    },
    telefono: {
        type: String,
        required: true,
        unique: true,
        trim: true,
        match: /^\d{9}$/
    },
    edad: {
        type: Number,
        min: 18,
        max: 120
    }
});
```

# Ejemplo de aplicación 1

- Definición del modelo
  - Un modelo es la abstracción que vamos a utilizar para realizar operaciones en la BD
  - Siempre que queramos realizar operaciones (insertar, eliminar, actualizar, etc) a MongoDB, debemos hacerlo a travez del modelo Contacto.

```
let Contacto = mongoose.model('contacto', contactoSchema);
```

# Ejemplo de aplicación 1

- Creamos el servidor Express, y lo ponemos a la escucha en el puerto 8080.
- Usamos un middleware body-parser, el cual analiza solo las solicitudes donde el encabezado Content-Type coincide con JSON.

```
// Servidor Express
let app = express();

// Middleware body-parser para peticiones POST y PUT
app.use(express.json());

// Puesta en marcha del servidor
app.listen(8080);
```

# Servicio de consulta de contactos

```
app.get('/contactos', (req, res) => {
    Contacto.find().then(resultado => {
        res.status(200)
           .send({ ok: true, resultado: resultado });
    }).catch (error => {
        res.status(500)
           .send({ ok: false, error: "Error obteniendo contactos"});
    });
});
```

# Servicio de consulta de contactos según su ID

```
app.get('/contactos/:id', (req, res) => {
    Contacto.findById(req.params.id).then(resultado => {
        if(resultado)
            res.status(200)
               .send({ ok: true, resultado: resultado });
        else
            res.status(400)
               .send({ ok: false,
                       error: "No se han encontrado contactos" });
    }).catch (error => {
        res.status(400)
           .send({ ok: false,
                   error: "Error buscando el contacto indicado" });
    });
});
```

# Servicio de inserción de contactos

```
app.post('/contactos', (req, res) => {

    let nuevoContacto = new Contacto({
        nombre: req.body.nombre,
        telefono: req.body.telefono,
        edad: req.body.edad
    });
    nuevoContacto.save().then(resultado => {
        res.status(200)
           .send({ok: true, resultado: resultado});
    }).catch(error => {
        res.status(400)
           .send({ok: false,
                  error: "Error añadiendo contacto"});
    });
});
```

Gracias al middleware tenemos en el cuerpo del objeto request (req) los datos enviados en la petición

# Servicio de modificación de contactos

```
app.put('/contactos/:id', (req, res) => {

    Contacto.findByIdAndUpdate(req.params.id, {
        $set: {
            nombre: req.body.nombre,
            telefono: req.body.telefono,
            edad: req.body.edad
        }
    }, {new: true}).then(resultado => {
        if (resultado)
            res.status(200)
               .send({ok: true, resultado: resultado});
        else
            res.status(400)
               .send({ok: false, error: "Contacto no encontrado"});
    }).catch(error => {
        res.status(400)
           .send({ok: false,
                  error:"Error actualizando contacto"});
    });
});
```

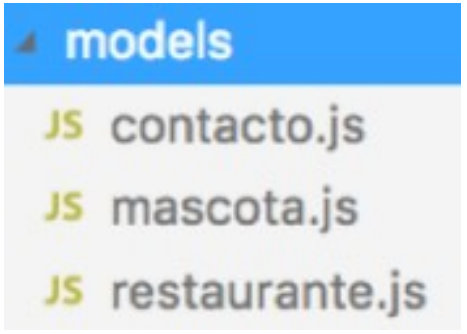
La opción new indica si queremos obtener como resultado el nuevo objeto modificado (true) o el antiguo antes de modificarse (false).

# Servicio de borrado de contactos

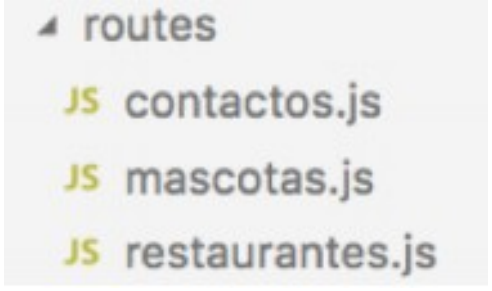
```
app.delete('/contactos/:id', (req, res) => {

    Contacto.findByIdAndRemove(req.params.id).then(resultado => {
        if (resultado)
            res.status(200)
               .send({ok: true, resultado: resultado});
        else
            res.status(400)
               .send({ok: false, error: "Contacto no encontrado"});
    }).catch(error => {
        res.status(400)
           .send({ok: false,
                  error:"Error eliminando contacto"});
    });
});
```

# Estructura de una API REST en EXPRESS




- **models:** se definen los modelos de las colecciones de datos




- **routes / controllers:** se define el código de los enrutadores. Se crea un archivo fuente para cada grupo de rutas

# Estructura de una API REST en EXPRESS

- En nuestro ejemplo de contactos, creamos una carpeta "**models**", donde definimos los archivos para los tres modelos de datos: **contacto.js** , **restaurante.js** y **mascota.js**.
  - En cada archivo definimos el esquema y el modelo (hay que incluir la librería mongoose en cada uno de ellos)

# models/contacto.js

```
const mongoose = require('mongoose');

// Definición del esquema de nuestra colección
let contactoSchema = new mongoose.Schema({
    nombre: {
        type: String,
        required: true,
        minlength: 1,
        trim: true
    },
    telefono: {
        type: String,
        required: true,
        unique: true,
        trim: true,
        match: /^\d{9}$/
    },
    edad: {
        type: Number,
        min: 18,
        max: 120
    }
});

// Asociación con el modelo
// (colección contactos)
let Contacto = mongoose.model(
    'contacto', contactoSchema);

module.exports = Contacto;
```

# models/mascota.js

```
const mongoose = require('mongoose');

// Mascotas
let mascotaSchema = new mongoose.Schema({
    nombre: {
        type: String,
        required: true,
        minlength: 1,
        trim: true
    },
    tipo: {
        type: String,
        required: true,
        enum: ['perro', 'gato', 'otros']
    }
});
```

```
// Modelo
let Mascota = mongoose.model(
    'mascota', mascotaSchema);

module.exports = Mascota;
```

# models/restaurante.js

```
const mongoose = require('mongoose');

// Restaurantes
let restauranteSchema = new mongoose.Schema({
    nombre: {
        type: String,
        required: true,
        minlength: 1,
        trim: true
    },
    direccion: {
        type: String,
        required: true,
        minlength: 1,
        trim: true
    },
    telefono: {
        type: String,
        required: true,
        unique: true,
        trim: true,
        match: /^\d{9}$/
    }
});

// Modelo
let Restaurante = mongoose.model(
    'restaurante', restauranteSchema);

module.exports = Restaurante;
```

# Rutas y enrutadores

- La gestión de contactos (alta / baja / modificación / consulta de contactos) se realiza mediante servicios englobados en una URI que empieza por **/contactos**.
- Para el caso de restaurantes y mascotas, utilizaremos las URIs **/restaurantes** y **/mascotas**, respectivamente.
- Hay que definir, por tanto, 3 enrutadores (un archivo fuente para cada grupo de rutas), dentro de “routes”.
  - En cada archivo, utilizaremos el modelo correspondiente de la carpeta "models" para poder manipular la colección asociada.

# routes/mascotas.js

```
const express = require('express');

let Mascota = require(__dirname + '/../models/mascota.js');
let router = express.Router();

// Servicio de listado
router.get('/', (req, res) => {
    Mascota.find().then(resultado => {
        res.status(200)
           .send({ok: true, resultado: resultado});
    }).catch (error => {
        res.status(500)
           .send({ok: false, error: "Error obteniendo mascotas"});
    });
});
```

Se define un objeto Router de Express para gestionar las rutas con un prefijo común o sobre una misma entidad. (antes era sobre la propia aplicación: objeto app).

Las rutas no hacen referencia a la URI /mascotas, sino que apuntan a una raíz /

# routes/mascotas.js

```
// Servicio de inserción
router.post('/', (req, res) => {
    let nuevaMascota = new Mascota({
        nombre: req.body.nombre,
        tipo: req.body.tipo
    });

    nuevaMascota.save().then(resultado => {
        res.status(200)
           .send({ok: true, resultado: resultado});
    }).catch(error => {
        res.status(400)
           .send({ok: false, error: "Error añadiendo mascota"});
    });
});
```

# routes/mascotas.js

```
// Servicio de borrado
router.delete('/:id', (req,res) => {
    Mascota.findByIdAndRemove(req.params['id'])
    .then(resultado => {
        if (resultado)
            res.status(200)
               .send({ok: true, resultado: resultado});
        else
            res.status(400)
               .send({ok: false, error: "Mascota no encontrada"});
    }).catch(error => {
        res.status(400)
           .send({ok: false, error: "Error borrando mascota"});
    });
});

module.exports = router;
```

# Rutas y enrutadores

- Definir los servicios GET, POST y DELETE para los restaurantes en el enrutador **routes / restaurantes.js**
- Para los servicios de contactos, adapta los de sesiones anteriores, copiándolos en el enrutador **routes/contactos.js** .

# La aplicación principal

- Cargar  las librerías y enrutadores
- Conectar con la base de datos
- Pone en marcha el servidor

# Aplicación estructurada en carpetas para una API REST completa sobre contactos, restaurantes y mascotas

```
// Librerías
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');

// Enrutadores
const mascotas = require(__dirname + '/routes/mascotas');
const restaurantes = require(__dirname + '/routes/restaurantes');
const contactos = require(__dirname + '/routes/contactos');

// Conexión con la BD
mongoose.connect(
    'mongodb://localhost:27017/contactos',
    {useNewUrlParser:true, useUnifiedTopology:true}
    );

// Servidor Express
let app = express();
```

# Aplicación estructurada en carpetas para una API REST completa sobre contactos, restaurantes y mascotas

```
// Middleware body-parser para peticiones POST y PUT
// Enrutadores para cada grupo de rutas
app.use(bodyParser.json());
app.use('/mascotas', mascotas);
app.use('/restaurantes', restaurantes);
app.use('/contactos', contactos);

// Puesta en marcha del servidor
app.listen(8080);
```

Los enrutadores se cargan como middleware, empleando app.use, especificando la ruta con la que se mapea cada enrutador.

Dentro de cada enrutador las rutas ya hacen referencia a esa ruta base que se les asigna desde el servidor principal, y por eso todas comienzan por / .