REALIZA UNA API REST CON NODE.JS

- PASOS QUE VAMOS A REALIZAR ANTES DE EMPEZAR:
  - INTALAR NODE.JS (LA VERSIÓN QUE SE ENCUENTRE EN ESE MOMENTO EN LTS)
  - DECARGAR MONGODB Y DESPUÉS DESCARGAR ROBO3T
  - DESCARGAR MYSQL COMUNNITY SERVER O EN SU DEFECTO XAMPP
  - DESCARGAR VISUAL STUDIO CODE
  - DESCARGAR ALGÚN CLIENTE DE BASE DE DATOS SQL (RECOMIENDO DESCARGAR DBEAVER)
  - DESCARGAR CLIENTE RESTFUL(POSTMAN)

1. **INICIAR UN NUEVO PROYECTO DESDE CERO CON NODE.JS**
   a. **Iniciar proyecto con npm init**
   b. **Cuando nos pregunte por main poner index.js**
   c. **Licencia MIT**
   d. **Terminar diciendo yes**
2. **INSTALAR LOS SIGUIENTES PAQUETES:**
   a. **Npm install express**
   b. **Npm install body-parser**
   c. **Npm install connect-multiparty**
   d. **Npm install mongoose**
   e. **Npm install nodemon - - save-dev**
   f. **Dentro del archivo package.json en el apartado scripts poner "start": "nodemon index.js"**
3. **Crear una nueva base de datos desde robo3T de mongo llamada colegio**
4. **Crear en el proyecto un archivo llamado app.js en el cual pondremos:**

```
'user strict';

var express = require('express');
var bodyParser = require('body-parser');

var app = express();

//cargar archivos ed rutas

//middlewares
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

//CORS

//rutas
app.get('/', (req, res) => {
  res.status(200).send({
    message: 'Hola mundo desde mi API REST'
  });
```

```
});

//exportar
module.exports = app;
```

5. **Crear un archivo llamado index.js y poner lo siguiente. Una vez terminado lanzar el comando npm start y comprobar en el navegador poniendo htpp://localhost:3700/**

```
'use strict';

var mongoose = require('mongoose');
var app = require('./app');
var port = 3700;

mongoose.Promise = global.Promise;
mongoose
  .connect('mongodb://localhost:27017/colegio')
  .then(msg => {
    console.log('Conexión establecida: ' + msg);

    //Creación del servidor
    app.listen(port, () => {
      console.log(
        'Servidor corriendo correctamente en la url: localhost:3700 '
      );
    });
  })
  .catch(err => console.log(err));
```

6. **Crear una nueva carpeta llamada models y dentro de ella un archive llamado alumno.js y dentro de ella escribir lo siguiente:**

```
'use strict';

var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var AlumnoSchema = Schema({
  nombre: String,
  apellido1: String,
  apellido2: String,
  edad: Number,
  asignaturas: [String]
});
```

```
module.exports = mongoose.model('Alumno', AlumnoSchema);
//con esto mongoose nos podra alumnos en lugar de Alumno y
//nos lo guardará como colección en la base de datos
```

7. **Creamos una nueva carpeta en el directorio raíz llamada controllers y dentro de ella creamos un archivo llamado alumno.js y escribimos lo siguiente**

```
'use strict';

var controller = {
  inicio: function(req, res) {
    return res.status(200).send({
      message: 'Soy el inicio'
    });
  },

  test: function(req, res) {
    return res.status(200).send({
      message: 'Soy el método test del controlador'
    });
  }
};

module.exports = controller;
```

8. **Volvemos al directorio raíz y ahora creamos una carpeta que se llame routes, en la cual creamos un archivo llamado alumno.js y escribimos lo siguiente:**

```
'use strict';

var express = require('express');
var AlumnoController = require('../controllers/alumno');

var router = express.Router();

router.get('/inicio', AlumnoController.inicio);
router.post('/test', AlumnoController.test);

module.exports = router;
```

a continuación, nos vamos al archivo app.js y cambiamos el contenido por (he dejado una errata)

```
'user strict';

var express = require('express');
var bodyParser = require(' body-parserito');

var app = express();

//cargar archivos de rutas
var alumno_routes = require('./routes/alumno');

//middlewares
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

//CORS


//rutas
app.use('/api', alumno_routes);

//exportar
module.exports = app;
```

después arrancamos el proyecto con npm start, y abrimos el programa postman y lanzamos peticiones a nuestros servicios para ver si todo funciona correctamente

9. **Vamos a crear el método para guardar alumnos**
    a. **En 'controllers/alumno.js' importamos el modelo de alumno y creamos una nueva función que se llamará guardarAlumno en la que pondremos toda la lógica y modificaremos además el modelo de alumno.js:**

```
guardarAlumno: function(req, res) {
    var alumno = new Alumno();

    var params = req.body;
    alumno.nombre = params.nombre;
    alumno.apellido1 = params.apellido1;
    alumno.apellido2 = params.apellido2;
    alumno.edad = params.edad;
    alumno.foto = null;

    alumno.save((err, alumnoGuardado) => {
      if (err)
        return res
          .status(500)
          .send({ message: 'Error al guardar el documento.' });
```

```
      if (!alumnoGuardado)
        return res
          .status(404)
          .send({ message: 'No se ha podido guardar el alumno' });

      return res.status(200).send({
        alumno: alumnoGuardado,
        message: 'Alumno guardado correctamente'
      });
    });
  }
```

Ahora modificamos el archivo del 'models/alumno.js' y lo dejamos del siguiente modo

```
var AlumnoSchema = Schema({
  nombre: String,
  apellido1: String,
  apellido2: String,
  edad: Number,
  foto: String
});
```

Ahora debemos de añadir al archivo 'routes/alumno.js' la ruta del controlador

```
router.post('/guardar-alumno', AlumnoController.guardarAlumno);
```

10. **DEVOLVER UN ALUMNO DE LA BASE DE DATOS**
    a. **Crearemos el método getAlumno en el archivo 'controllers/alumno.js' y después añadiremos la ruta y la importación del método al 'routes/alumno.js'**

```
getAlumno: function(req, res) {
    var alumnoId = req.params.id;


    if (alumnoId == null) {
      return res.status(404).send({ message: 'El alumno no existe.' });
    }


    Alumno.findById(alumnoId, (err, alumno) => {
      if (err)
```

```
        return res.status(500).send({ message: 'Error al devolver los
datos' });


      if (!alumno)
        return res.status(404).send({ message: 'El alumno no existe.' });


      return res.status(200).send({
        alumno
      });
    });
  }
```

Añadimos la ruta

```
router.get('/alumno/:id', AlumnoController.getAlumno);
```

y probamos desde el postman que funcione correctamente

11. **DEVOLVER LISTADO DE ALUMNOS ordenados por su edad ordenados de mayor a menor (para ello antes de todo insertaremos varios alumnos en nuestra base de datos) y crear su nueva ruta en el controlador de alumnos (he dejado una errata)**

```
getAlumnos: function(req, res) {
    AlumnitosDe.find({})
      .sort('-edad')
      .exec((err, alumnos) => {
        if (err)
          return res
            .status(500)
            .send({ message: 'Error al devolver los datos' });

        if (!alumnos)
          return res
            .status(404)
            .send({ message: 'No hay alumnos para mostrar' });

        return res.status(200).send({ alumnos });
      });
  }
```

Crear ruta en el controlador

```
router.get('/alumnos', AlumnoController.getAlumnos);
```

12. **Ahora Crearemos un método y su controlador para poder actualizar un alumno**

```
actualizarAlumno: function(req, res) {
    var alumnoId = req.params.id;
    var update = req.body;

    Alumno.findByIdAndUpdate(
      alumnoId,
      update,
      { new: true },
      (err, alumnoUpdated) => {
        if (err)
          return res.status(500).send({ message: 'Error al actualizar'
});

        if (!alumnoUpdated)
          return res
            .status(404)
            .send({ message: 'No se ha podido actualizar el alumno!' });

        return res.status(200).send({
          alumno: alumnoUpdated,
          message: 'Alumno actualizado con éxito'
        });
      }
    );
  }
```

Creamos su controlador (usaremos el método http PUT porque vamos a actualizar)

```
router.put('/alumno/:id', AlumnoController.actualizarAlumno);
```

probamos desde postman que funcione correctamente

13. **CREAMOS UN MÉTODO PARA BORRAR UN ALUMNO Y SU RUTA EN EL CONTROLADOR**

```
deleteAlumno: function(req, res) {
    var alumnoId = req.params.id;

    Alumno.findByIdAndRemove(alumnoId, (err, alumnoRemoved) => {
      if (err)
        return res
          .status(500)
          .send({ message: 'No se ha podido borrar el alumno' });

      if (!alumnoRemoved)
        return res
          .status(404)
          .send({ message: 'No se puede eliminar ese alumno' });

      return res.status(200).send({
        alumno: alumnoRemoved,
        message: 'Alumno borrado correctamente'
      });
    });
  }
```

Creamos la ruta en el controlador y probamos desde Postman

```
router.delete('/alumno/:id', AlumnoController.deleteAlumno);
```

14. **AHORA VAMOS A CREAR UN MÉTODO PARA SUBIR IMÁGENES, PARA ELLO VAMOS A USAR EL connect-multiparty QUE LO TUVIMOS QUE INSTALAR COMO PAQUETE DE DEPENDENCIAS DE NODE.JS AL PRINCIPIO, PARA ELLO PRIMERO DEBEREMOS DE IR AL ARCHIVO DONDE CONFIGURAMOS LAS RUTAS DE NUESTROS CONTROLADORES E IMPORTAR EL SIGUIENTE MODULO**

```
var multipart = require('connect-multiparty');
var multipartMiddlewate = multipart({uploadDir: './uploads'});
```

ahora nos creamos una carpeta en la raíz de nuestro api rest llamada 'uploads', ahí será donde se guarden todos nuestros archivos de imágenes de los alumnos. Una vez hecho esto nos vamos a nuestro archivo donde tenemos las rutas de los controladores y escribimos nuestra ruta

```
router.post(
  '/upload-image/:id',
  multipartMiddlewate,
  AlumnoController.uploadImage
);
```

Y ahora creamos el método uploadImage en nuestro archivo donde tenemos los métodos del controlador 'controllers/alumno.js', pero antes importamos la librería 'fs':

```
var fs = require('fs');
```

y ahora creamos el método

```
uploadImage: function(req, res) {
    var alumnoId = req.params.id;
    var fileName = 'Imagen no subida...';

    if (req.files) {
      var filePath = req.files.image.path;
      var fileSplit = filePath.split('\\');
      var fileName = fileSplit[1];
      var extSplit = fileName.split('.');
      var fileExt = extSplit[1];

      if (
        fileExt.toLowerCase() == 'png' ||
        fileExt.toLowerCase() == 'jpg' ||
        fileExt.toLowerCase() == 'jpeg' ||
        fileExt.toLowerCase() == 'gif'
      ) {
        Alumno.findByIdAndUpdate(
          alumnoId,
          { image: fileName },
          { new: true },
          (err, alumnoUpdated) => {
            if (err)
              return res
                .status(200)
                .send({ message: 'La imagen no se ha subido' });

            if (!alumnoUpdated)
              return res.status(404).send({
                message:
```

```
                  'El alumno no existe y no se ha podido asignar la
imagen'
              });

            return res.status(200).send({
              files: alumnoUpdated,
              message:
                'Se ha insertado la imagen correctamente en el alumno con
id ' +
                alumnoId
            });
          }
        );
      } else {
        fs.unlink(filePath, err => {
          return res
            .status(400)
            .send({ message: 'La extensión del archivo no es valida' });
        });
      }
    } else {
      return res.status(200).send({
        message: fileName
      });
    }
  }
```

15. **AHORA VAMOS A IMPLEMENTAR EL CORS PARA SOLUCIONAR EL PROBLEMA O POSIBLES PROBLEMAS AL REALIZAR PETICIONES AJAX A NUESTRO BACKEND PARA PERMITIR EL ACCESO CRUZADO ENTRE LOS DOMINIOS DEL FRONT AL BACKEND (SI NO SABES BIEN QUE ES TE ANIMO A QUE INVESTIGUES UN POCO SOBRE LOS ACCESOS A CORS), PARA ELLO NOS VAMOS A NUESTRO ARCHIVO APP.JS Y EN LA PARTE DONDE TENEMOS PUESTO CORS AÑADIMOS EL SIGUIENTE CÓDIGO**

```
//CORS
app.use((req, res, next) => {
  res.header('Access-Control-Allow-Origin', '*');
  res.header(
    'Access-Control-Allow-Headers',
    'Authorization, X-API-KEY, Origin, X-Requested-With, Content-Type,
Accept, Access-Control-Allow-Request-Method'
  );
  res.header('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT,
DELETE');
  res.header('Allow', 'GET, POST, OPTIONS, PUT, DELETE');
  next();
});
```