ProjectManager | MERN

Instalación y dependencias | BACKEND

1. Crear un servidor con Express
    - *OPCIONAL: Añadiendo la propiedad "type" con valor "module" en el package.json se puede habilitar el uso de sintasis ESM en lugar de CommonJS*
    - Refactorizar el código, de modo que sea funcional a la arquitectura API REST
    - Instalar las siguientes dependencias (algunas ya vienen incluídas con *express-generator*):
        - "bcryptjs": "^2.4.3",
        - "concurrently": "^7.6.0",
        - "cors": "^2.8.5",
        - "debug": "~2.6.9",
        - "dotenv": "^16.0.3",
        - "express": "~4.16.1",
        - "http-errors": "~1.6.3",
        - "jsonwebtoken": "^9.0.0",
        - "mongoose": "^6.9.0",
        - "morgan": "~1.9.1",
        - "nodemailer": "^6.9.0"

Configurar la conexión con MongoDB

1. Crear la Base de Datos
    - Crear una cuenta en MongoDB
    - Descargar e instalar Mongo Compass
    - *OPCIONAL: Instalar la extensión MongoDB for VS Code*
    - Copiar y pegar el link de conexión ej: mongodb+srv://root:root@cluster0.hvchqdf.mongodb.net/test
2. Instalar en el ORM Mongoose
    - Enlaces de interés:
        - ¿Qué es Mongoose?
        - Introducción a Mongoose
3. Configuar la conexión

```
    const connectDB = async () => {
        try {
            mongoose.set('strictQuery',false);
            const connection = await mongoose.connect(process.env.DB_CONNECTION,{
                useNewUrlParser: true,
                useUnifiedTopology : true
            });

            const url =
`${connection.connection.host}:${connection.connection.port}`;
            console.log(`MongoDB connected in ${url}`)
```

```
        } catch (error) {
            console.log(`error: ${error.message}`);
            process.exit(1); //obliga a terminar todos los procesos
        }
    }
```

Notas:

- `useNewUrlParser` MongoDB ha dejado de usar su analizador de cadenas de conexión actual. Debido a que este es un cambio importante, agregaron el indicador `useNewUrlParser` para permitir que los usuarios recurran al analizador antiguo si encuentran un error en el analizador nuevo. Ver más
- `useUnifiedTopology`, por defecto en false. Establézcalo en verdadero para optar por usar el nuevo motor de administración de conexiones del controlador MongoDB. Debe establecer esta opción en true , excepto en el caso improbable de que le impida mantener una conexión estable.Ver más

Crear los modelos

1. Crear el modelo de **User** en la carpeta `/models`.
    - *OPCIONAL: Instalar la extensión Mongo Snippets for Node-js*
    - *OPCIONAL*: Utilizar el snippets: **!mdbgum** para crear el modelo*

```
const mongoose = require('mongoose');
const {hash, compare} = require('bcryptjs');

var userSchema = new mongoose.Schema({
    name:{
        type:String,
        required:true,
        trim:true,
    },
    email:{
        type:String,
        required:true,
        trim : true,
        unique:true,
    },
    password:{
        type:String,
        required:true,
        trim:true,
    },
    token:{
        type:String,
    },
    checked :{
        type: Boolean,
        default : false,
    },

},
```

```
    {
        timestamps : true
    });

    /* se ejecuta antes de guardar el documento */
    userSchema.pre('save', async function (next) {
        if(!this.isModified('password')){ //evita volver a hashear el password
cuando este no ha sido modificado
            next()
        }
        this.password = await hash(this.password, 10);
    });

    /* puedo crear más métodos */
    userSchema.methods.checkedPassword = async function(password){
        return await compare(password, this.password)
    }

    module.exports = mongoose.model('User', userSchema);
```

2. Crear el modelo de **Project** en la carpeta `/models`.
    - *OPCIONAL: Instalar la extensión Mongo Snippets for Node-js*
    - *OPCIONAL*: Utilizar el snippets: **!mdbgum** para crear el modelo*

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

var projectSchema = new mongoose.Schema({
    name:{
        type:String,
        required:true,
        trim:true,
    },
    description:{
        type:String,
        required:true,
        trim:true,
    },
    dateExpire:{
        type:Date,
        default:Date.now(),
    },
    client:{
        type:String,
        required:true,
        trim:true,
    },
    createdBy:{
        type:mongoose.Schema.Types.ObjectId,
        ref: 'User',
    },
    collaborators :[
```

```
        {
            type:mongoose.Schema.Types.ObjectId,
            ref: 'User',
        }
    ],
},{
    timestamps : true,
});

module.exports = mongoose.model('Project', projectSchema);
```

3. Crear el modelo de **Task** en la carpeta `/models`.
    - *OPCIONAL: Instalar la extensión Mongo Snippets for Node-js*
    - *OPCIONAL*: Utilizar el snippets: **!mdbgum** para crear el modelo*

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

var taskSchema = new mongoose.Schema({
    name:{
        type:String,
        required:true,
        trim : true,
    },
    description:{
        type:String,
        required:true,
        trim:true,
    },
    state:{
        type:Boolean,
        default:false,
    },
    dateExpire:{
        type:Date,
        required:true,
        default : Date.now()
    },
    priority : {
        type : String,
        required:true,
        enum : ['Baja','Media','Alta'], //opciones cerradas
    },
    project : {
        type:mongoose.Schema.Types.ObjectId,
        ref: 'Project',
    }
},{
    timestamps : true
});

module.exports = mongoose.model('Task', taskSchema);
```

Crear los controladores

1. Controlador de Autenticación `authController.js`

```
module.exports = {
    register : async (req,res) => {
        try {

            return res.status(201).json({
                ok : true,
                msg :'Usuario Registrado'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en REGISTER'
            })
        }

    },
    login : async (req,res) => {
        try {
            return res.status(200).json({
                ok : true,
                msg :'Usuario Logueado'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en LOGIN'
            })
        }

    },
    checked : async (req,res) => {
        try {
            return res.status(201).json({
                ok : true,
                msg :'Usuario Checkeado'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en CHECKED'
            })
        }

    },
```

```
    sendToken : async (req,res) => {
        try {
            return res.status(200).json({
                ok : true,
                msg :'Token enviado'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en SEND-TOKEN'
            })
        }
    },
    verifyToken : async (req,res) => {
        try {
            return res.status(200).json({
                ok : true,
                msg :'Token verificado'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en VERIFY-TOKEN'
            })
        }
    },
    changePassword : async (req,res) => {
        try {
            return res.status(200).json({
                ok : true,
                msg :'Password actualizado'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en CHANGE-PASSWORD'
            })
        }
    },
}
```

2. Controlador de Proyectos `projectsController.js`

```
module.exports = {
    list : async (req,res) => {
        try {
            return res.status(200).json({
                ok : true,
                msg :'Lista de Proyectos'
```

```
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en PROJECTS-LIST'
            })
        }

    },
    store : async (req,res) => {
        try {
            return res.status(201).json({
                ok : true,
                msg :'Proyecto guardado'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en STORE-PROJECT'
            })
        }

    },
    detail : async (req,res) => {
        try {
            return res.status(200).json({
                ok : true,
                msg :'Detalle del Proyecto'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en PROJECT-DETAIL'
            })
        }

    },
    update : async (req,res) => {
        try {
            return res.status(201).json({
                ok : true,
                msg :'Proyecto actualizado'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en PROJECT-UPDATE'
            })
        }
    },
```

```
    remove : async (req,res) => {
        try {
            return res.status(200).json({
                ok : true,
                msg :'Proyecto eliminado'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en PROJECT-REMOVE'
            })
        }
    },
    addCollaborator : async (req,res) => {
        try {
            return res.status(200).json({
                ok : true,
                msg :'Colaborador agregado'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en COLLABORATOR-ADD'
            })
        }
    },
    removeCollaborator : async (req,res) => {
        try {
            return res.status(200).json({
                ok : true,
                msg :'Colaborador eliminado'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en COLLABORATOR-
REMOVE'
            })
        }
    },

}
```

3. Controlador de Tareas `tasksController.js`

```
module.exports = {
    list : async (req,res) => {
        try {
            return res.status(200).json({
```

```
                ok : true,
                msg :'Lista de Tareas'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en TASKS-LIST'
            })
        }
    },
    store : async (req,res) => {
        try {
            return res.status(201).json({
                ok : true,
                msg :'Tarea guardado'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en STORE-TASK'
            })
        }

    },
    detail : async (req,res) => {
        try {
            return res.status(200).json({
                ok : true,
                msg :'Detalle de la Tarea'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en TASK-DETAIL'
            })
        }

    },
    update : async (req,res) => {
        try {
            return res.status(201).json({
                ok : true,
                msg :'Tarea actualizada'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en TASK-UPDATE'
            })
        }
```

```
    },
    remove : async (req,res) => {
        try {
            return res.status(200).json({
                ok : true,
                msg :'Tarea eliminado'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en TASK-REMOVE'
            })
        }
    },
    changeState : async (req,res) => {
        try {
            return res.status(200).json({
                ok : true,
                msg :'Tarea completada'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en CHANGE-STATE'
            })
        }
    },

}
```

4. Controlador de Usuarios `usersController.js`

```
module.exports = {
    profile : async(req,res) => {
        try {
            return res.status(200).json({
                ok : true,
                msg :'Perfil de Usuario'
            })
        } catch (error) {
            console.log(error);
            return res.status(error.status || 500).json({
                ok : false,
                msg : error.message || 'Upss, hubo un error en PERFIL'
            })
        }
    }
}
```

Crear los enrutadores

1. Enrutador de Autenticación `auth.js`

```
const express = require('express');
const router = express.Router();

const {register,verifyToken,login,changePassword,checked,sendToken} =
require('../controllers/authController')

/* /api/auth */

router
    .post('/register', register )
    .post('/login',login)
    .get('/checked',checked)
    .post('/send-token',sendToken)
    .route('/reset-password')
        .get(verifyToken)
        .post(changePassword)

module.exports = router;
```

2. Enrutador de Proyectos `projects.js`

```
const express = require('express');
const router = express.Router();

const { list, store, detail, update, remove, addCollaborator, removeCollaborator }
= require('../controllers/projectsController')

/* /api/projects */

router
    .route('/')
        .get(list)
        .post(store)
router
    .route('/:id')
        .get(detail)
        .put(update)
        .delete(remove)
router
    .get('/collaborator/add', addCollaborator)
    .delete('/collaborator/remove',removeCollaborator)

module.exports = router;
```

3. Enrutador de Tareas `tasks.js`

```
const express = require('express');
const router = express.Router();

const { list, store, detail, update, remove, changeState, } =
require('../controllers/tasksController')

/* /api/tasks */

router
    .route('/')
        .get(list)
        .post(store)
router
    .route('/:id')
        .get(detail)
        .put(update)
        .delete(remove)
router
    .post('/change-state/:id', changeState)

module.exports = router;
```

4. Enrutador de Usuarios `users.js`

```
const express = require('express');
const router = express.Router();

const {profile} = require('../controllers/usersController')

/* /api/users */
router.get('/', profile);

module.exports = router;
```

5. Vincular los enrutadores en `app.js`

```
  /* RUTAS */
app
  .use('/api/auth',require('./routes/auth'))
  .use('/api/users',require('./routes/users'))
  .use('/api/projects',require('./routes/projects'))
  .use('/api/tasks',require('./routes/tasks'))
```