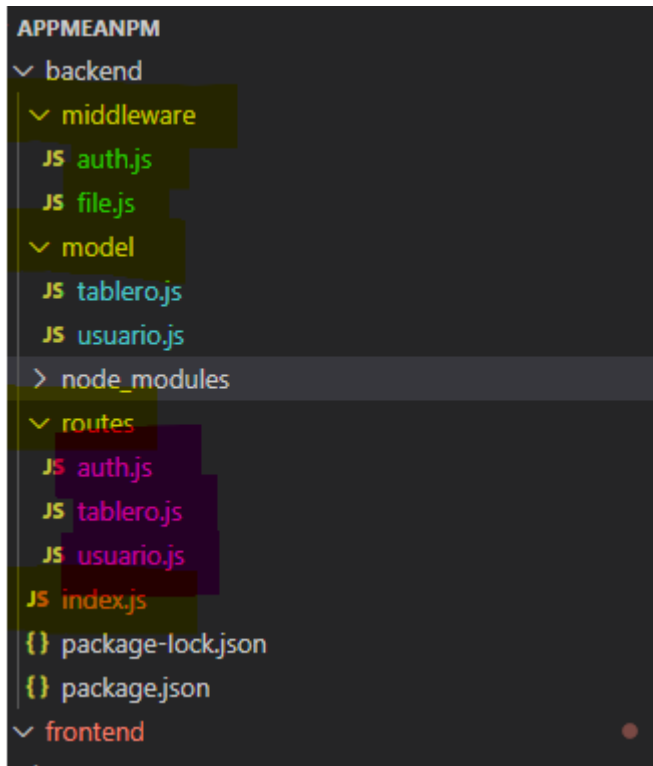
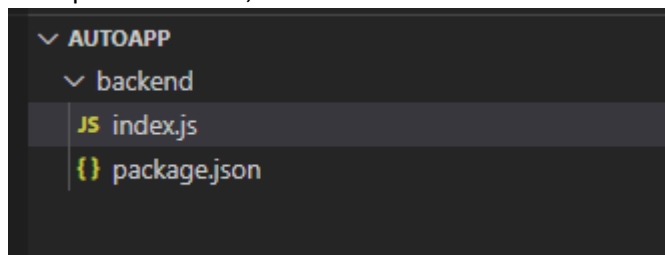


Taller NodeJS/Express MongoDB Angular

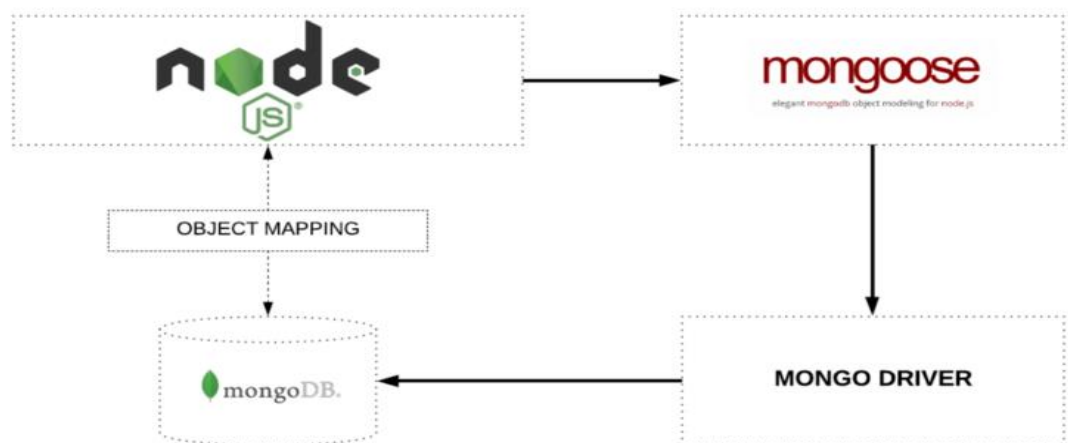
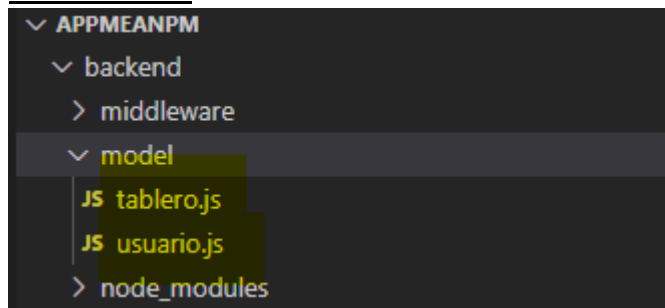
- Mockup
- Model DataBase
- Controlador
- Front (Angular)



1. Crear la carpeta **APPMEANPM** y el subfolder **backend** (inicializar proyecto NodeJS)
2. **APPMEANPM\backend>npm init** (se genera automáticamente archivo package.json en la carpeta backend , adicionalmente creamos el archivo index.js.



3. Instalar **express** al proyecto `APPMEANPM\backend` `>npm install express`
4. Instalar librería **mongoose** para administrar MONGODB `APPMEANPM\backend >npm install mongoose`
5. Instalar **jsonwebtoken** para encriptar información `APPMEANPM\backend >npm install jsonwebtoken`
6. Creamos un subcarpeta llamada **model** con los archivos `usuario.js` `tablero.js` módulos que van a guardar las colecciones en MONGODB en la carpeta backend. () similar Entity Java Persistence API



7. Crear el modelo de datos en el archivo `usuario.js`, los módulos internos y el esquema del usuario.

```
backend > model > JS usuario.js > ...
1 // cuando registran desde angular
2 //modulos internos node
3 const mongoose = require("mongoose");
4 const jwt = require("jsonwebtoken");
5 // Esquema
6 const esquemaUsuario = new mongoose.Schema({
7   nombre: {
8     type: String,
9   },
10  cedula: {
11    type: String,
12  },
13
14  edad: {
15    type: Number,
16  },
17  correo: {
18    type: String,
19  },
20  pass: {
21    type: String,
22  },
23  fecha: {
24    type: Date,
25    default: Date.now,
26  }
27 });
```

8. Generar en el esquema de `usuario.js` el método de encriptación `JsonWebToken` con la exportación del modulo con el esquema del modelo base de datos.

```

// Generar el JWT

esquemaUsuario.methods.generateJWT=function() {
return jwt.sign({
  _id:this.id,
  nombre:this.nombre,
  correo:this.correo
}, "clave");

};
// exportamos modulo
const Usuario= mongoose.model("usuario",esquemaUsuario);

module.exports.Usuario=Usuario;
module.exports.esquemaUsuario=esquemaUsuario;

```

9. **CONTROLADOR** Realizar la rutas **URL de la API**(como se va a registrar un usuario **CRUD**) para la colección **usuario.js** crear la subcarpeta routers y exportar el modulo **router**(propiedad de EXPRESS)

```

backend > routes > JS usuario.js > router.post("/") callback
1
2 // API a funcionar nos llega por metodo POST desde angular los campos que registro el usua
3 const express= require("express");
4 const router= express.Router();
5
6 //modulos internos creados por el desarrollador
7
8 const {Usuario}= require("../model/usuario");
9
10 //usa el modelo de model/usuario
11
12 // cuando completa un formulario desde angular lo envian por POST
13 router.post("/", async (req, res) =>{
14
15 // espera a que se cumpla un proceso la validacion si correo existe DB
16 let usuario= await Usuario.findOne({correo:req.body.correo})
17
18 if (usuario) return res.status(400).send("El correo ya esta registrado");
19 // mapeo para registrar lo que llego por parte del usuario desde el Front(angular)
20 y guardar en MONGODB
21
22 usuario= new Usuario({
23   nombre: req.body.nombre,
24   cedula: req.body.cedula,
25   edad: req.body.edad,
26   correo: req.body.correo,
27   pass: req.body.pass

```

10. En la subcarpeta **routes** crear el archivo **auth.js** para validar y crear token de usuario y exportar el modulo router(propiedad de EXPRESS)

```
backend > routes > JS auth.js > router.post("/") callback
1 // Modulos de Node
2 const express = require("express");
3 const router = express.Router();
4
5
6 // Modulos internos para el sistema de login traemos el modelo usuario
7 const { Usuario } = require("../model/usuario");
8 // Ruta
9 router.post("/", async (req, res) => {
10 // Validamos que el correo exista
11 const usuario = await Usuario.findOne({ correo: req.body.correo });
12 // Si el correo no existe
13 if (!usuario)
14 | return res.status(400).send("Correo o contraseña no son validos");
15 // si el pass no existe
16 if (usuario.pass !== req.body.pass)
17 | return res.status(400).send("Correo o contraseña no son validos");
18 // cuando una persona se loguea Generamos un JWT para que index.js lo pueda utilizar
19 // lo importamos desde el model de usuario
20 const jwtToken = usuario.generateJWT();
21 res.status(200).send({ jwtToken });
22 });
23 module.exports = router;
```

11. En la subcarpeta **middleware** crear el archivo **auth.js** para el traspaso de información de usuario con el token de validación creando una función.

12. En la subcarpeta **middleware** crear el archivo **file.js** para el traspaso de información y subir los sticker(archivos imagen) instalamos libreria **MULTER** **APPMEANPM** **backend** **npm i multer.**

```

backend > middleware > JS file.js > cargarArchivo
1  //modulos de node
2  const multer = require("multer");
3  //ruta del directorio donde quedaran los archivos
4  const directorio = "./public";
5  //DiskStorage
6  const storage = multer.diskStorage({
7    destination: (req, file, cb) => {
8      cb(null, directorio);
9    },
10   filename: (req, file, cb) => {
11     const filename =
12       Date.now() + "-" + file.originalname.toLowerCase().split(".").join("-");
13     cb(null, filename)
14   },
15 });
16 //Cargar archivos
17 const cargarArchivo = multer({
18   storage: storage,
19   fileFilter: (req, file, cb) => {
20     console.log(file.mimetype)
21     if (

```

13. CONTROLADOR Realizar la rutas URL de la API (como se va crear (POST), listar (GET), actualizar (PUT) borrar (DELETE) actividad con o sin imagen (CRUD) para la colección **tablero.js** crear la subcarpeta routers y exportar el modulo router(propiedad de EXPRESS)

```

▼ backend
  > middleware
  > model
  > node_modules
  ▼ routes
    JS auth.js
    JS tablero.js
    JS usuario.js

```

backend > routes > JS tablero.js > ...

```
49 //Crear actividad con imagen
50 router.post("/cargarArchivo", cargarArchivo.single("sticker"), auth, async (req, res) => {
51   const url = req.protocol + "://" + req.get("host");
52   // Validamos si existe el usuario
53   const usuario = await Usuario.findById(req.usuario._id);
54   // si el usuario no existe
55   if (!usuario) return res.status(400).send("El usuario no existe en BD");
56   // Si existe el usuario continuamos el proceso
57   let rutaImagen = null;
58   if (req.file.filename) {
59     rutaImagen = url + "/public/" + req.file.filename;
60   } else {
61     rutaImagen = null;
62   }
63   // Guardar la actividad con imagen en BD
64   const tablero = new Tablero({
65     idUsuario: usuario._id,
66     nombre: req.body.nombre,
67     descripcion: req.body.descripcion,
68     sticker: rutaImagen,
69     estado: req.body.estado,
70   });
71   // Enviamos resultado
72   const result = await tablero.save();
73   res.status(200).send(result);
74 }
75 }:
```

13. En el archivo modulo index.js que es la creación de nuestra APP con el pool de la conexión de la Base de Datos en MongoDB.

```
backend > JS index.js > ...
1  // index.js invoca la ruta del controlador routes/usuario.js
2
3  const express= require("express");
4  const mongoose= require("mongoose");
5  const cors= require("cors")
6
7  // traer o importar el usuario del modulo routes
8  const usuario= require("../routes/usuario");
9  const auth= require("../routes/auth");
10 const tablero= require("../routes/tablero");
11
12 // crear nuestra app
13 const app =express();
14 // todo la comunicación va a ser formato JSON
15 app.use(cors());
16 app.use(express.json());
17 // indicamos cual es la ruta de la API cuando se este registrando un usuario
18 app.use("/api/usuario/", usuario);
19 app.use("/api/auth/", auth);
20 app.use("/api/tablero/", tablero);
21
```

```
// puerto para ejecutar nuestro aplicacion servidor

const port = process.env.PORT || 3000;
app.listen(port, ()=> console.log("Escuchando en el puerto :" +port));

// creamos la conexion con la base de datos MONGODB
mongoose
.connect("mongodb://localhost/scrum",{
  // utilizar 4 parametros por defecto para que funcionen bien
  useUrlParser: true,
  useFindAndModify: false,
  useCreateIndex: true,
  useUnifiedTopology: true,
})
.then(()=> console.log("Conexión a MongoDB: online"))
.catch((error) => console.log("Conexión a MongoDB: offline") );
// se recomienda utilizar .then and .catch para conexiones.
```


14. Se conecta a mongod mongo ejecutar compass

```
C:\Users\APTO>mongod
{"t":{"$date":"2020-10-27T20:07:20.777-05:00"},"s":"I", "c":"CONTROL", "id":23285, "ctx":
disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2020-10-27T20:07:20.900-05:00"},"s":"W", "c":"ASIO", "id":22601, "ctx":
layer configured during NetworkInterface startup"}
{"t":{"$date":"2020-10-27T20:07:20.947-05:00"},"s":"I", "c":"NETWORK", "id":4648602, "ctx":
FastOpen in use."}
{"t":{"$date":"2020-10-27T20:07:21.060-05:00"},"s":"I", "c":"STORAGE", "id":4615611, "ctx":
oDB starting", "attr":{"pid":3576, "port":27017, "dbPath":"C:/data/db/", "architecture":"64-bit"}
{"t":{"$date":"2020-10-27T20:07:21.096-05:00"},"s":"I", "c":"CONTROL", "id":23398, "ctx":
et operating system minimum version", "attr":{"targetMinOS":"Windows 7/Windows Server 2008 R2
```

15. Se conecta mongo

```
\Users\APTO\Documents>mongo
MongoDB shell version v4.4.1
Connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("a2de8ec9-9421-462d-9d40-6c40bd948cad") }
MongoDB server version: 4.4.1
-
The server generated these startup warnings when booting:
  2020-10-16T15:45:30.176-05:00: ***** SERVER RESTARTED *****
  2020-10-16T15:45:37.409-05:00: Access control is not enabled for the database. Read and write a
onfiguration is unrestricted
-
-
  Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

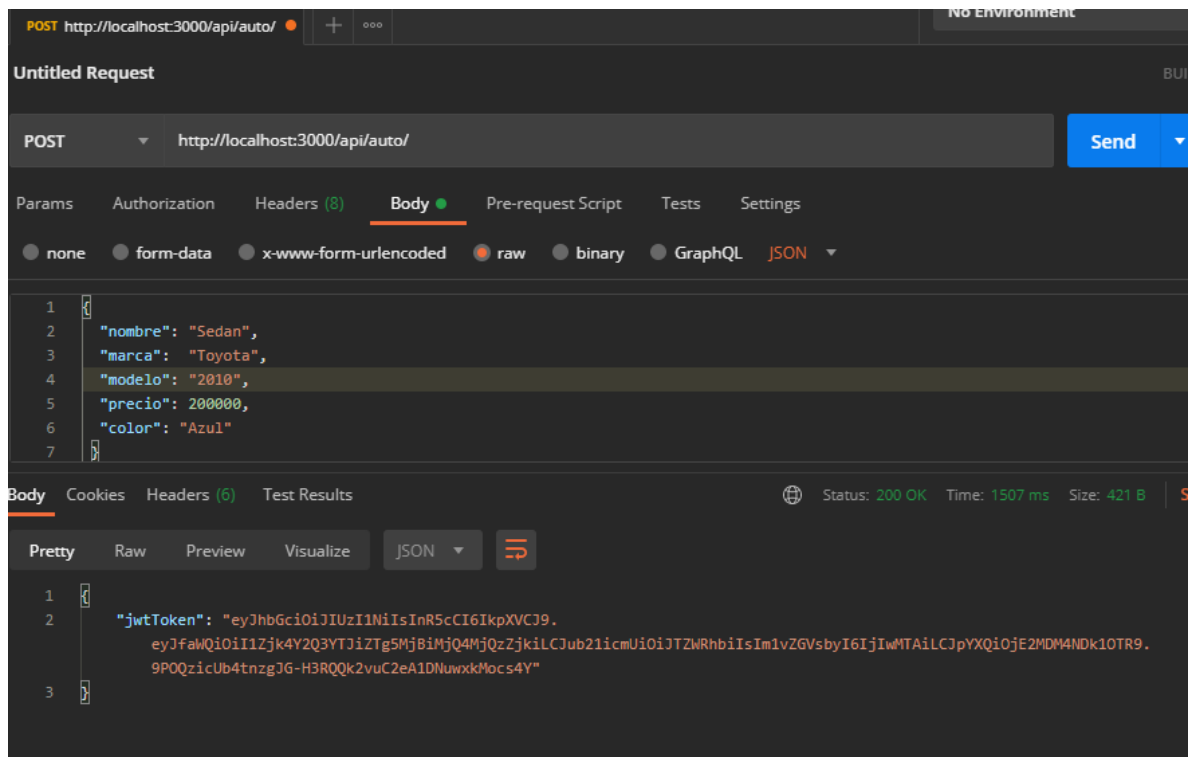
  The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

  To enable free monitoring, run the following command: db.enableFreeMonitoring()
  To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
```

16. Desplegar API con conexión a MongoDB AutoApp\backend>node index.js

```
at internal/main/run_main_module.js:17:147
C:\Users\APTO\Documents\CHALLENGES\AutoApp\backend>node index.js
Escuchando en el puerto 3000
Conexión a MongoDB online
[]
```

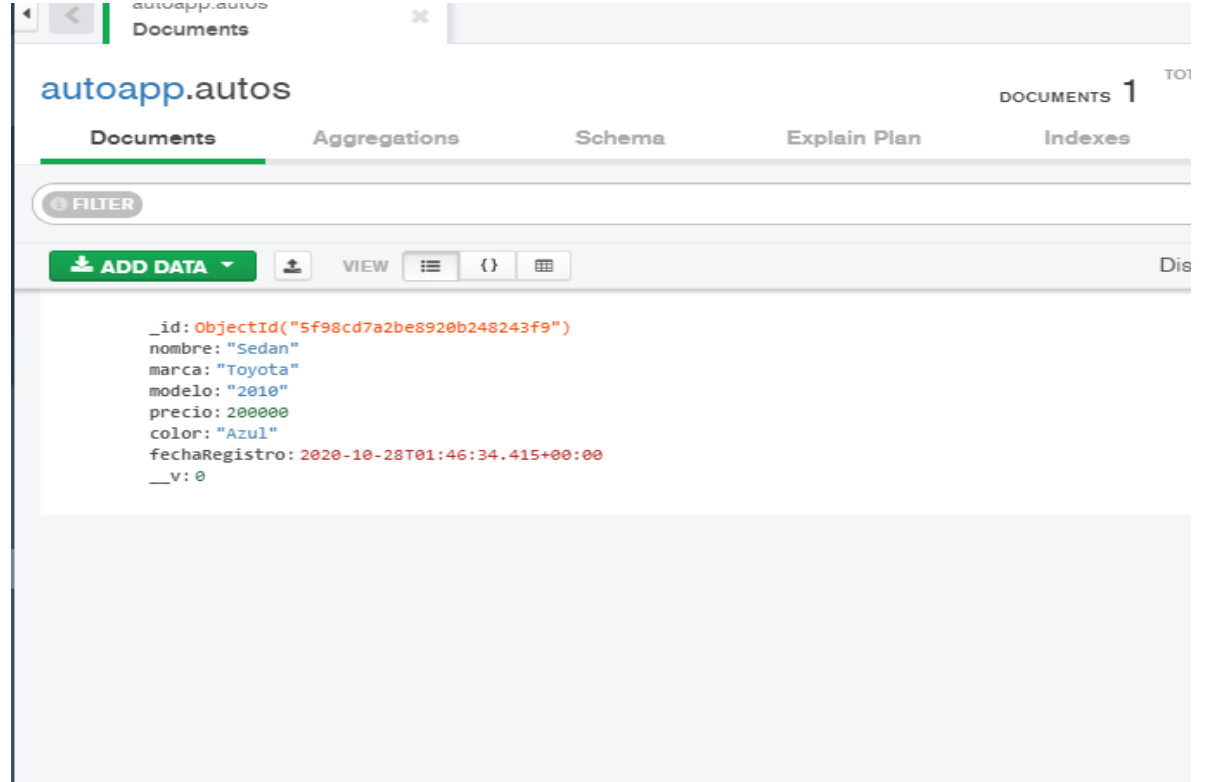
17. Postman convierte el documento auto de la colección autoapp a jsonwebtoken



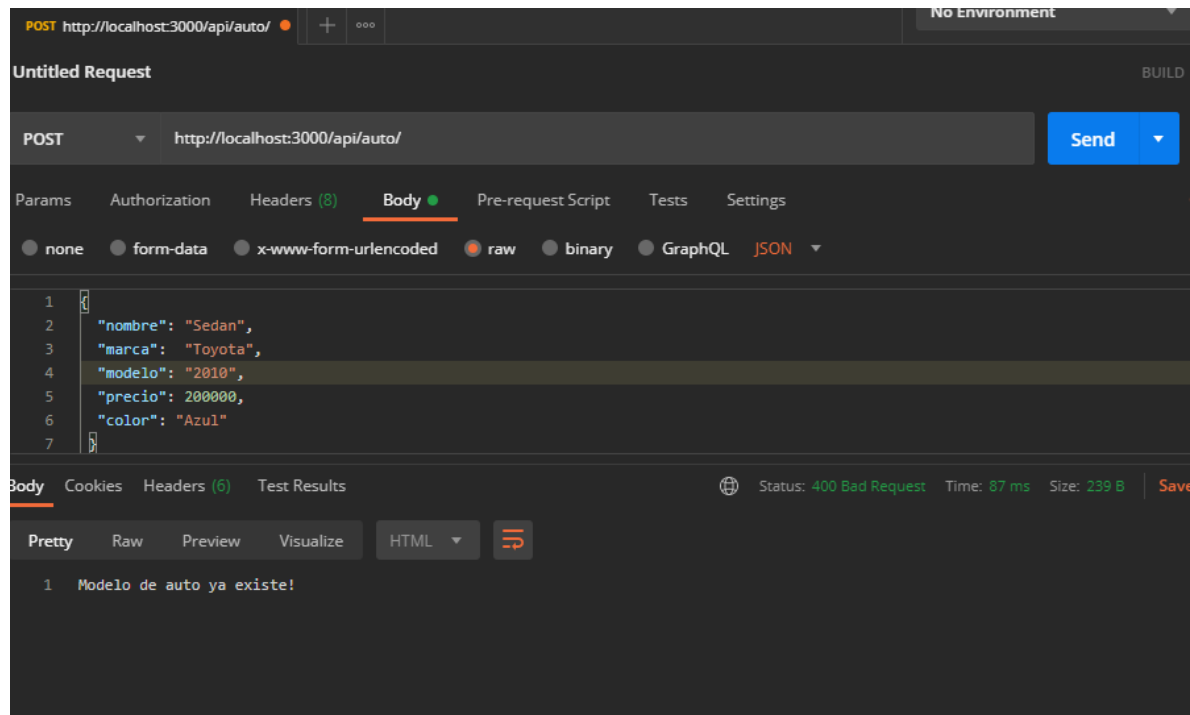
EN **ANGULAR** método en el constructor del archivo Auth de servicio

```
registroUsuario(usuario){  
  return this.http.post<any>(this.registroUrl, usuario);  
}  
  
loginUsuario(usuario){  
  return this.http.post<any>(this.loginUrl, usuario);  
}
```

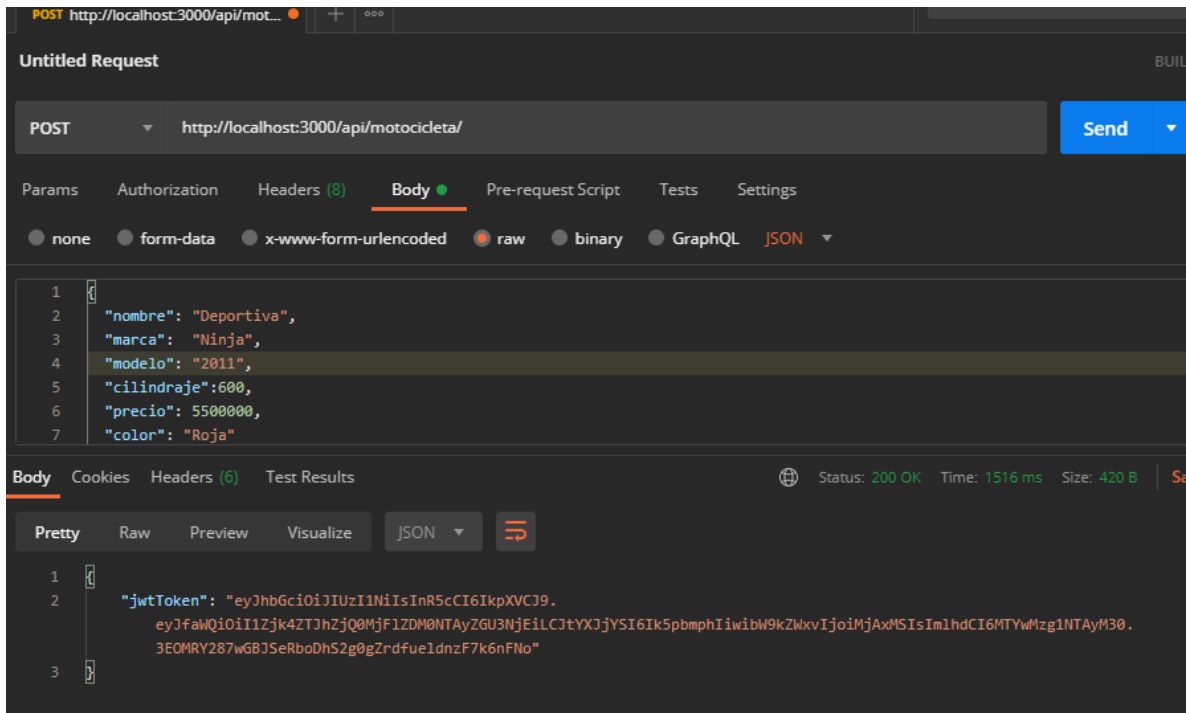
14. Se crea desde la app la base de datos MongoDB con la colección auto automáticamente



15. Enviar nuevamente el mismo modelo 2010 se genera mensaje de “error”



16. Postman convierte el documento auto de la colección autoapp a jsonwebToken



17. Se crea desde la app la base de datos MongoDB con la colección auto automáticamente

Documents

autoapp.motocicletas

DOCUMENTS 1

Documents Aggregations Schema Explain Plan Indexes

FILTER

ADD DATA

VIEW

Dis

```
_id: ObjectId("5f98e2af4421ed34502de761")
nombre: "Deportiva"
marca: "Ninja"
modelo: "2011"
cilindraje: 2011
precio: 5500000
color: "Roja"
fecha: 2020-10-28T03:17:03.059+00:00
__v: 0
```

Para empezar nuestro frontend con ANGULAR comunicar las diferentes servidores y urls de nuestro backend con el frontend en nuestro caso localhost:3000 en backend con localhost:4000 en frontend instalamos libreria CORS APPMEANPM\backend>npm i cors.

En el index.js en el backend los configuramos

```
backend > JS index.js > ...
1 // index.js invoca la ruta del controlador routes/usuario.js
2
3 const express= require("express");
4 const mongoose= require("mongoose");
5 const cors= require("cors")
6
7 // traer o importar el usuario del modulo routes
8 const usuario= require("../routes/usuario");
9 const auth= require("../routes/auth");
10 const tablero= require("../routes/tablero");
11
12 // crear nuestra app
13 const app =express();
14 // todo la comunicación va a ser formato JSON
15 app.use(cors());
16 app.use(express.json());
```

INSTALAR PROYECTO EN ANGULAR

\APPMEANPM>ng new frontend --routing

CREACIÓN DE COMPONENTES ANGULAR

ng g c menú

ng g c login

ng g c registro

ng g c tablero/crear

ng g c tablero/listar

SISTEMA DE NAVEGACION app routing .module

ANGULAR tiene herramienta para optimizar FLEX y GRID LAYOUT

Instalar APPMEANPM\frontend>npm install -s @angular/flex-layout @angular/cdk

found 1 high severity vulnerability (Para arreglar vulnerabilidad)

APPMEANPM\frontend>npm audit fix

<https://material.io/resources/icons/?style=baseline> (Ejemplos de iconos a utilizar)

menú.componente.css

menú.componente.html (asociamos los iconos al menú)

```
menu.component.css  <> menu.component.html ●
ontend > src > app > menu > <> menu.component.html > mat-toolbar.menu
1  <mat-toolbar class="menu">
2
3      <mat-icon aria-hidden="false" aria-label="event">event</mat-icon>
4
5      <ul>
6          <a routerLink="crearActividad">Crear actividad</a>
7      </ul>
8      <span class="spacerIcon"></span>
9      <ul>
10         <mat-icon aria-hidden="false" aria-label="fact_check">fact_check</mat-icon>
11         <a routerLink="listarActividad">Lista de actividades</a>
12     </ul>
13     <span class="spacer"></span>
14     <ul>
15         <mat-icon aria-hidden="false" aria-label="account_circle"
16             >account_circle</mat-icon>
17         >
18         <a routerLink="login">Login</a>
19     </ul>
20     <ul>
21         <mat-icon aria-hidden="false" aria-label="account_box"
22             >account_box</mat-icon>
23         >
24         <a routerLink="registro">Registro</a>
25     </ul>
26 </mat-toolbar>
```

Registrar un usuario registro.component.ts y que se pueda loguear para generar JsonWebToken

App.module importamos material/card'; form-field'; /input';

```

frontend > src > app > TS app.module.ts > 🚧 AppModule
15   import {MatCardModule} from '@angular/material/card';
16   import {MatFormFieldModule} from '@angular/material/form-field';
17   import {MatInputModule} from '@angular/material/input';
18
19
20
21
22   @NgModule({
23     declarations: [
24       AppComponent,
25       MenuComponent,
26       LoginComponent,
27       RegistroComponent,
28       CrearComponent,
29       ListarComponent
30     ],
31     imports: [
32       BrowserModule,
33       AppRoutingModule,
34       BrowserAnimationsModule,
35       MatToolbarModule,
36       MatIconModule,
37       MatButtonModule,
38       MatCardModule,
39       MatFormFieldModule,
40       MatInputModule,
41

```

Se trabaja con formulario se `import {FormsModule} from '@angular/forms';`

Se adiciona al `imports: [FormsModule,`

Para utilizar javascript consumir el backend comunicar con el backend

APPMEANPM\frontend> ng g s service/auth

Para comunicarnos con backend API REST necesitamos importar

```
import {HttpClientModule} from '@angular/common/http';
```

LOS SERVICIOS VAN EN PROVIDER

AuthService Provee un servicio y se configura en `providers: [AuthService],`

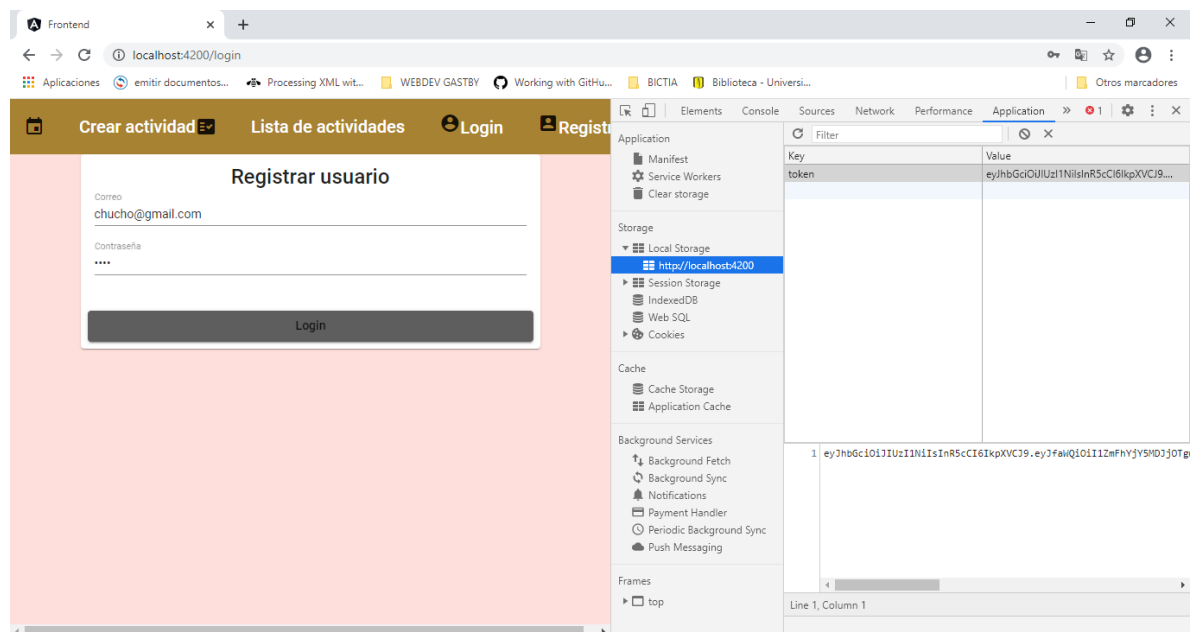
Boton de registrar usuario en

```

menu.component.css  TS app.module.ts  TS auth.service.ts
content > src > app > service > TS auth.service.ts > AuthService
1  import { Injectable } from '@angular/core';
2  import { HttpClient } from '@angular/common/http';
3
4  @Injectable({
5    providedIn: 'root'
6  })
7  export class AuthService {
8
9    private registroUrl='http://localhost:3000/api/usuario/';
10   private loginUrl= 'http://localhost:3000/api/auth/';
11
12   //constructor lo primero que se ejecuta antes de lo que construye toda la class
13   constructor(private http: HttpClient) { }
14
15   registroUsuario(usuario){
16     return this.http.post<any>(this.registroUrl, usuario);
17   }
18
19   loginUsuario(usuario){
20     return this.http.post<any>(this.loginUrl, usuario);
21   }
22 }

```

Cuando usuario se loguea se genera un JasonWebToken que se guarda con un local storage



Apenas me logue veo **mi lista de actividades** , cuando me registro me debe llevar **al login**.(2rutas)

Login component.ts

```
//subscribe es leer la respuesta del observable. Es un req que se envia desde angular a backend
login(){
  // en el res viene un jasonwebtoken es la respuesta de backend a Angular
  this.auth.loginUsuario(this.loguear).subscribe((res) =>{
    console.log(res);
    // apenas hago el login guardo res.jwtToken(info encriptada) en un localStorage
    localStorage.setItem('token',res.jwtToken);
    // cuando usuario se encuentra logueado con token lo lleva a la pagina ruta listarActividad
    this.router.navigate(['/listarActividad']);
  },
  (err) => console.log(err)
);
```

Si escriben cualquier url <http://localhost:4200/crearActividad> al usuario le debe salir **login**

Necesitamos un sistema que nos indique que el usuario este logeado, ANGULAR tiene una librería **GUARD** para proteger si tenemos un token permita ver las URL

Instalar **\APPMEANPM\frontend>ng g guard guard/auth**

Nuestro sistema de guardia para proteger URL , ejemplo registro y login son publicas (no necesitan guard)

proteger url

```
const routes: Routes = [
  {
    //una ruta de navegación 'vacío ' lo primero que carga
    path: '',
    component: LoginComponent,
    pathMatch: 'full'
  },
  {
    path: 'listarActividad',
    component: ListarComponent,
    canActivate: [AuthGuard],
  },
  {
    path: 'crearActividad',
    component: CrearComponent,
    canActivate: [AuthGuard],
  },
  {
    path: 'login',
    component: LoginComponent,
  },
]
```

En Backend con el POSTMAN para subir una imagen usuario logueado se pega token en Autorization con el bearer.

Se debe instalar un servicio `APPMEANPM\frontend>ng g s service/tokenInterceptor`

<https://material.angular.io/components/expansion/api>

en APP module la importamos

import { MatExpansionModule } from '@angular/material/expansion'; y lo declaramos en import

Los cambios de los botones de estado se hacen desde [listar.component.html](#)

```
// es un css accordion display ejemplo estilo cards se vean pequeños en la
mitad
    display: flex;
    justify-content: center;
    align-items: center;
<div class="accordion_display"> los divs no organizan los espacios
// css accordion size

    <div class="accordion_size">
        Mat accordion propiedad multi="true" multiples acordiones
        <mat-accordion multi="true" *ngIf="lista.length"> si hay una lista es
un array con 2 actividades lista.length 2 actividades para pintar los
accordions
        <mat-expansion-panel *ngFor="let lista of lista" hideToggle="false">
recurso a un for para iterar la lista y mostrar las actividades y cada una
pintarla hideToggle de material
        <mat-expansion-panel-header>

            <mat-panel-title>
                <span class="text_center">{{ lista.nombre }}</span>
Interpolación el front muestra el valor de la variable
                <span class="spacer"></span> spacer aplicar un espacio
3 BOTONES
<button
    background-color: #3cb44b;
    color: #ffffff;
    class="btDone" Dependiendo el estado cambia el color
    *ngIf="lista.estado == 'terminada'" botón aparece si lista
estado dice terminada
    mat-raised-button
    >
        {{ lista.estado }}
</button>
<button
    class="btInProgress"
    *ngIf="lista.estado == 'iniciada'"
    mat-raised-button
    >
        {{ lista.estado }}
</button>
<button
    class="btToDo"
    *ngIf="lista.estado == 'asignada'"
    mat-raised-button
```

```

    >
    {{ lista.estado }}
  </button>
</mat-panel-title>
Botones Parte de la Descripción de nuestra actividad
</mat-expansion-panel-header>
<mat-panel-description>
  {{ lista.descripcion }}
</mat-panel-description>
Nos permite hacer acciones tener eventos
<mat-action-row>
  <button
    class="btSelectTodo" Me va aparecer la palabra con el estado
    mat-button
    (click)="cambiarEstado(lista, 'asignada')"
Click para activar cambiarEstado recibo lista la actividad (hacer login) que
estamos ahora y el estado que quiero cambiar a asignada
  >
    Asignada
  </button>
  <button
    class="btSelectInProgress"
    mat-button
    (click)="cambiarEstado(lista, 'iniciada')"
Click para activar cambiarEstado recibo lista la actividad (hacer login) que
estamos ahora y el estado que quiero cambiar a iniciada
  >
    Iniciada
  </button>
  <button
    class="btSelectDone"
    mat-button
    (click)="cambiarEstado(lista, 'terminada')"
Click para activar cambiarEstado recibo lista la actividad (hacer login) que
estamos ahora y el estado que quiero cambiar a terminada
  >
    Terminada
  </button> el botón sería un icono
  <mat-icon para darle a un icono un evento
    color="primary"
    (click)="eliminar(lista)".ed
    aria-label="delete_forever"
  >delete_forever</mat-icon

```

```
        >  
        </mat-action-row>  
    </mat-expansion-panel>  
</mat-accordion>  
</div>  
</div>
```