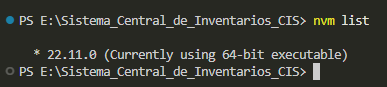
# Creación de proyecto CIS

## Instalar NVM para Windows

Para gestionar versiones de Node.js en diferentes proyectos se debe descargar e instalar “**nvm-setup.exe**”.

* Url de descarga 🡪 <https://github.com/coreybutler/nvm-windows/releases>
* Para ver la lista de versiones instaladas de Node.js o la que se está usando actualmente.

#### **nvm list**



* Si quieres que una versión específica se use automáticamente, puedes configurarla como predeterminada.

#### **nvm use 22.11.0**

Texto

Descripción generada automáticamente

## Instalación de NodeJS

Para instalar NodeJS se debe seguir los siguientes pasos:

* Descargar versión de NodeJS desde la URL 🡪 <https://nodejs.org/dist>
* Url de interés <https://nodejs.org/en/about/previous-releases>

Texto

Descripción generada automáticamente

## Instalación de Angular y Creación de Proyecto

Angular es uno de los Framewok de JavaScript más utilizados para el desarrollo Frontend. Su objetivo es aumentar las aplicaciones basadas en MVC para agilizar el desarrollo y pruebas en el Frontend, antes de su instalación crear carpeta en la raíz del proyecto para este caso la llamaremos **CIS\_Appv1**

* Url de interés

🡪 <https://angular.io/guide/versions>

🡪 <https://angular.dev/reference/versions>

🡪 <https://www.npmjs.com/package/@angular/cli?activeTab=versions>

* Desde la carpeta raíz del proyecto “**cisfront**” instala o actualiza Angular CLI y el Framewok de Angular (@angular/core) en sus versiones más recientes, última versión estable “**latest**” o una versión específica.

Instalación local

#### **npm install @angular/cli@latest @angular/core@latest –save-dev**

#### **npm install @angular/cli@18.2.11 @angular/core@18.2.12 –save -dev**

Instalación global

#### **npm install -g @angular/cli@latest @angular/core@latest**

#### **npm install -g @angular/cli@18.2.11 @angular/core@18.2.12**

* Desde la carpeta creada para el proyecto **CIS\_Appv1** se ejecuta el siguiente comando para crear el proyecto
* Para ver si existen dependencias desactualizadas

***npm outdated***

* Para actualizar algún paquete desactualizado

***npm install <paquete>@latest***

#### **ng new cisfront**

Texto

Descripción generada automáticamente

Actualización local

#### **ng update @angular/cli @angular/core**

#### **ng update @angular/cli@17.x @angular/core@17.x**

Ver versiones

(*Se debe hacer desde la carpeta del proyecto creada para que nos traiga toda la información*)

#### **ng --version**

#### **ng version**

* Al actualizar las versiones de cli o core, puede que nos pida como hacer la migración, pulsamos “**space”** para seleccionar las que necesitemos o a para seleccionar todas las migraciones, por último, “**enter”** para continuar.

Texto

Descripción generada automáticamente con confianza media

* Validar versión dentro del proyecto

#### **ng version**

Texto

Descripción generada automáticamente

* Agregar o validar en el archivo “**package.json**”, para que nos permite ejecutar el comando ng desde el terminal.

Una pantalla de un video juego

Descripción generada automáticamente con confianza media

## Versiones en Angular

La propiedad "**version**" en el **package.json** sigue la convención estándar de versiones semánticas (Semantic Versioning o SemVer). El formato típico es X.Y.Z, donde:

* X: Número de versión principal (cambios incompatibles con versiones anteriores).
* Y: Número de versión secundaria (nuevas características compatibles con versiones anteriores).
* Z: Número de versión de parche (correcciones de errores compatibles con versiones anteriores).

Texto

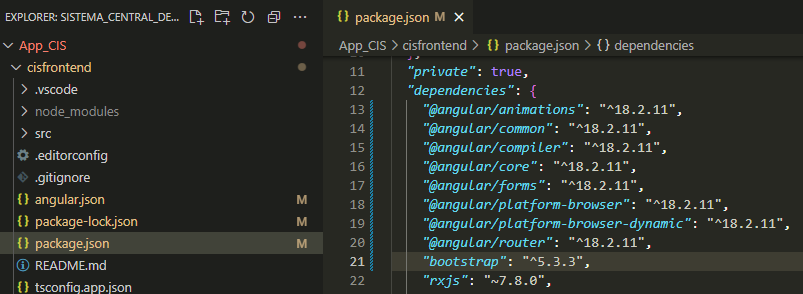
Descripción generada automáticamente

## Instalación Bootstrap y configuración

Desde el terminal y dentro de la carpeta del proyecto de angular **cisfront** ejecutar el siguiente comando.

#### **npm i bootstrap**

* Para validar la instalación y versión del **Bootstrap** instalado podemos verlo en el archivo **package.json** directamente en la carpeta de Angular.



* En el archivo **angular.json** en la sección de **build** y en la opción de **styles** se debe agregar el repositorio de estilos de Bootstrap y en la opción de **scripts** se debe agregar el repositorio de Bootstrap de animaciones para poder utilizarlos en el proyecto.

Permite que los estilos de bootstrap se apliquen al proyecto

#### **"node\_modules/bootstrap/dist/css/bootstrap.css"**

Agrega tanto el JavaScript de Bootstrap como las dependencias necesarias para que los componentes funcionen correctamente, incluye componentes de bootstrap como jQuery y Popper.js

#### **"node\_modules/bootstrap/dist/js/bootstrap.bundle.js"**

Texto

Descripción generada automáticamente

* Videos, canciones, tipos de letra, imágenes se ubican dentro de la carpeta "**assets**" de “**src**” y debe estar en el archivo "**angular.json**" en --> "**assets**": **["src/favicon.ico", "src/assets"]**, o se puede crear otra carpeta dentro de “**assets**” y en el archivo “**angular.json**” se debe agregar dicha ruta en el campo “**assets**”.

Captura de pantalla de computadora

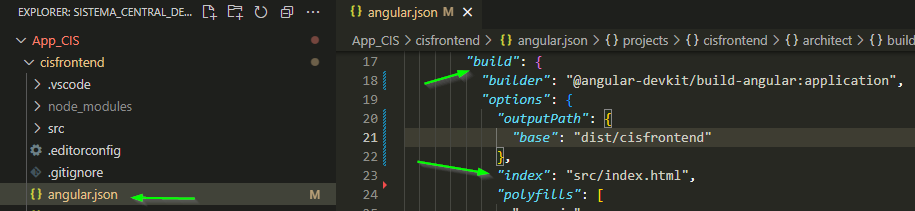
Descripción generada automáticamente

* En la parte “**scripts**” del archivo "**angular.json**" se configura o se traen todos los archivos “**JavaScript**” que se necesiten tener de forma global.
* Dentro del archivo "**angular.json**" en la opción "**architect**" --> "**outputPath**": "**dist/cisfront**", es el lugar donde va hacer la compilación.

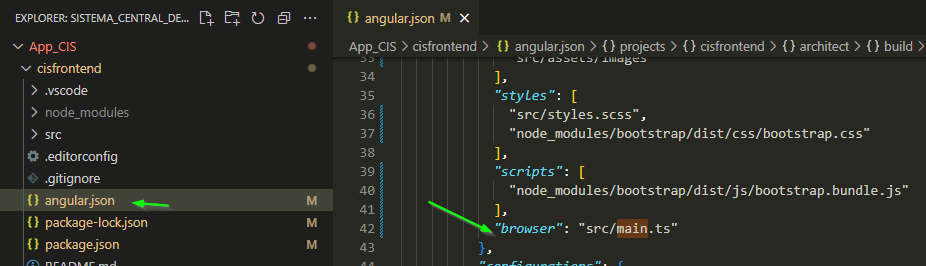
Captura de pantalla de computadora

Descripción generada automáticamente

* Dentro del archivo "**angular.json**" en la opción "**architect**" --> "**index**": "**src/index.html**", // Página de inicio que va cargar apenas ingresemos.



* Dentro del archivo "**angular.json**" en la opción "**architect**" --> "**browser**": "**src/main.ts**", es el archivo main por defecto del proyecto.



* Dentro del archivo "**angular.json**" en la opción "**root**" --> "**styles**": se puede configurar el scss global para los estilos.

## Ejecución del proyecto

Para ejecutar el proyecto desde el terminal se debe ejecutar desde la carpeta raíz de angular con cualquiera de los siguientes comandos:

#### **ng serve**

#### **npm start**

* Luego del anterior comando y su correcta compilación se debe ingresar a la ULR <http://localhost:4200> desde esta página podemos ver más información de interés para su manejo y aprendizaje.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

## Creación de módulos

Para la estructura del proyecto primero creamos los modulos y dentro de estos si creamos los componentes

#### **ng generate module /modules/cis --routing**

## Creación de componentes

Un componente se crea dentro de un módulo si este le pertenece o si no le pertenece se crea afuera, ejemplo una barra de navegación, un componente tiene:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

#### **ng generate component /modules/cis/index**

Texto

Descripción generada automáticamente

## Redireccionar index por defecto (Routing)

Para redireccionar el **index por defecto** a otro componente, debemos crear primero el componente al que queremos redirigirlo, en este caso lo llamamos **index** y se encuentra dentro de nuestro modulo llamado **mundochic**.

* Llamamos nuestro componente root dentro del index global del proyecto.

Texto

Descripción generada automáticamente

* En el archivo **app-routes.ts** se debe agregar la siguientes líneas en la sección de **routes**, una para redirigir el **index** hacia nuestro nuevo componente que se encuentra en **chic/index** y otra donde importamos el módulo llamado **chic** donde se encuentra el componente llamado **index**.

Captura de pantalla de un celular

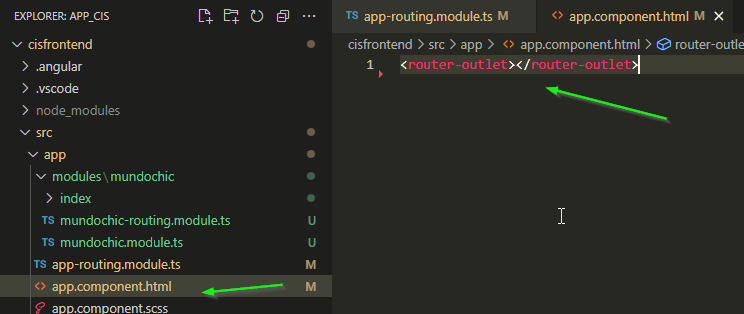
Descripción generada automáticamente

* Lo siguiente es en nuestro archivo **routing.module.ts** de nuestro modulo llamado **chic**, importamos el componente **index** que tenemos en nuestro modulo y en la sección de **routes** agregamos el componente para tener el routing.

Captura de pantalla de computadora

Descripción generada automáticamente

* Por último, para que funcione el routing debemos llamar **<router-outlet></router-outlet>** que nos permite mostrar diferentes componentes en función a la **URL** actual y esto lo hacemos dentro de nuestro **app.component.html**



## Navegación entre componentes (Router)

Permite crear aplicaciones de una sola página (SPA) dinámicas y eficientes. En lugar de recargar toda la página cada vez que el usuario navega a una nueva sección, el router de Angular reemplaza el contenido de una sección específica de la página, creando una experiencia de usuario más fluida y similar a la de una aplicación de escritorio.

Es decir, permite definir rutas a los diferentes componentes de tu aplicación. Al hacer clic en un enlace o botón, el router carga el componente asociado a esa ruta, actualizando la vista del usuario sin recargar toda la página.

* Importamos la librería en el componente donde tenemos nuestro enlace o botón que vamos a redirigir, en el constructor declaramos el router y creamos una función en este caso llamada login donde indicaremos la ruta a la que queremos acceder.

Texto

Descripción generada automáticamente

* Seguido de lo anterior debemos llamar esta función en nuestro enlace o botón para que al dar click, este se redirija a la ruta indicada.

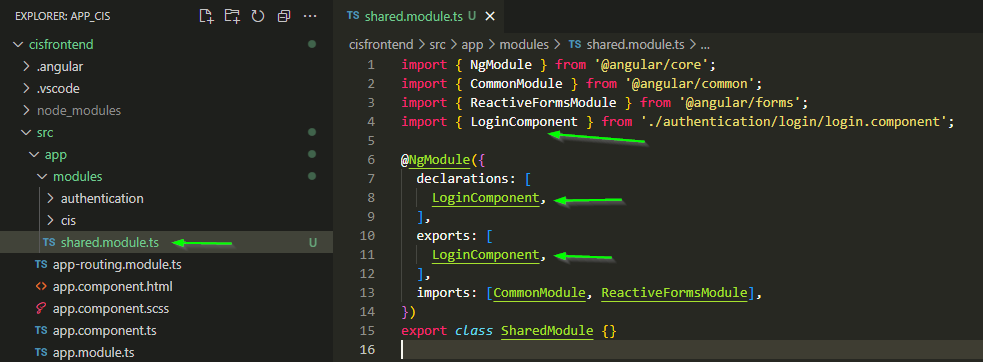
Texto

Descripción generada automáticamente

## Componentes compartidos

Si tienes un componente (como un menú) que quieres utilizar en diferentes partes de tu aplicación, crear un módulo compartido es una excelente práctica.

* Crea un nuevo archivo llamado **shared.module.ts** en tu carpeta de módulos (**modules/shared.module.ts**), allí importamos el componente a compartir y se declara en las opciones de **NgModul**e y **exports**, esto es útil y una buena practica como por ejemplo para un barra de menú.



* Luego, modifica el archivo **module.ts** de tus componentes donde necesites importar el **shared.module.ts** en lugar de importar y declarar directamente el componente compartido.

Texto

Descripción generada automáticamente

Al hacer estos cambios, importas y declaras el componente MenuComponent solo una vez en el módulo compartido y luego lo exportas para que pueda ser utilizado en otros módulos. Esto evita conflictos de declaración duplicada y te permite utilizar el componente en varios módulos sin problemas.

### Nota

Cada componente que se quiera compartir se debe agregar al **shared.module.ts** y eliminar del **app.component.ts**

## Modales

Un modal en Angular es una ventana emergente que se superpone al contenido principal de una aplicación web, proporcionando una interfaz para realizar acciones específicas sin que el usuario tenga que abandonar la página actual

ng add @angular/material

Texto

Descripción generada automáticamente

## Versiones de Software

Algo común es realizar el manejo de versiones mediante 3 números: X.Y.Z y cada uno indica una cosa diferente:

* El primero (X) se le conoce como versión mayor y nos indica la versión principal del software. Ejemplo: 1.0.0, 3.0.0
* El segundo (Y) se le conoce como versión menor y nos indica nuevas funcionalidades. Ejemplo: 1.2.0, 3.3.0
* El tercero (Z) se le conoce como revisión y nos indica que se hizo una revisión del código por algún fallo. Ejemplo: 1.2.2, 3.3.4

Ahora que conocemos el significado de cada número, viene una pregunta importante: ¿cómo sabemos cuándo cambiarlos y cuál cambiar?

Versión mayor o X, cuando agreguemos nuevas funcionalidades importantes, puede ser como un nuevo módulo o característica clave para la funcionalidad.

Versión menor o Y, cuando hacemos correcciones menores, cuando arreglamos un error y se agregan funcionalidades que no son cruciales para el proyecto.

Revisión o Z, cada vez que entregamos el proyecto.

### Versión por fecha.

En algunos necesitamos saber exactamente la fecha en que se publicó el software, entonces podremos utilizar el manejo de versiones por fecha. Este tiene muchas variaciones, se puede tener diferente orden del año, mes y día.

Ejemplo: 1.2.3.1543 donde 15 es el año 2015, 4 es el mes y 3 el día, como ya mencioné anteriormente se podrían tener diferentes acomodos y formatos: 1.2.3.4315 o 1.2.3.201543, 1.2.3.1534

Texto

Descripción generada automáticamente

## Formularios reactivos

* En el module compartido “**shared.module.ts**” importar lo siguiente y agregarlos a la sección de **imports**.



Texto

Descripción generada automáticamente con confianza media

* Importamos el modulo compartido en el modulo al que pertenece nuestro formulario reactivo, en este caso el formulario lo tenemos en el componente login que se encuentra dentro del módulo de autenticación.

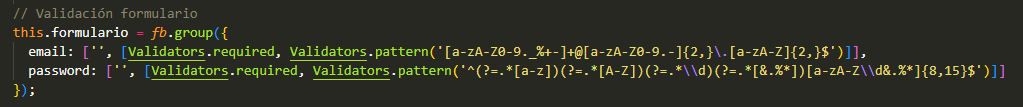


Imagen que contiene Texto

Descripción generada automáticamente

* En el componente donde se quiere tener el formulario reactivo, en este caso en login se debe tener lo siguiente:





* En el html login, se agrega el “**formGroup**” y los **formControlName** para los campos que queramos hacer reactivos.



Escala de tiempo

Descripción generada automáticamente

* Desactivar el botón del formulario hasta que se cumplan las validaciones



* Agregar evento click al botón en el html, en este caso se llama login y debe estar creado en el componente su redirección



Texto

Descripción generada automáticamente

## Conexión Frontend con Backend

En el proyecto de angular, para este caso creamos un módulo llamado “**home**” y dentro de este debemos crear un componente, que es donde se va a redireccionar una vez iniciemos sesión

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Luego de lo anterior seguir los siguientes pasos:

* Crear servicio dentro del módulo que se está realizando, en este caso autenticación.

Texto

Descripción generada automáticamente

* En el módulo de autenticación donde creamos el servicio anterior, debemos importar el **HttpClientModule**.



Texto

Descripción generada automáticamente

* En el servicio anteriormente creado “**security.service.ts**” se debe inyectar el servicio de **http**.

Texto

Descripción generada automáticamente

* Se debe crear un método inicio de sesión, en este caso se va a llamar **login** en el servicio creado anteriormente de seguridad.

Texto

Descripción generada automáticamente

* Consumir método anteriormente creado “**login**” en **security.service.ts** desde el componente login.



Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

* Crear método en el componente login

Texto

Descripción generada automáticamente

### Nota

Recordar que para que el routing funcione importar en el **app.module.ts** lo siguiente:



Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

# Instalación Loopback

Ejecutar el siguiente comando desde cualquier parte:

*npm install -g @loopback/cli@5.2.4*

*npm install -g @loopback/cli*

(instala la versión más actual)

Puede generar error al ejecutar comandos de loopback por que no esta habilitado la ejecución de script, se deben habilitar.

*Set-ExecutionPolicy RemoteSigned -Scope CurrentUser*

Validar que este en RemoteSigned

*Get-ExecutionPolicy -Scope CurrentUser*

## Creación de proyecto con Loopback

Ejecutar “**lb4 app**” y seguir los pasos para su creación, este creará el proyecto de Loopback y este debe quedar en la carpeta raíz del proyecto y al mismo nivel que el proyecto de Angular.

## Ejecución del proyecto

Para ejecutar el proyecto desde el terminal se debe ejecutar desde la carpeta raíz de loopback con el siguiente comando:

#### npm start

Captura de pantalla de un celular

Descripción generada automáticamente

Para corregir el error de comando desconocido al ejecutar el comando de ejecución, se debe instalar el npm con el siguiente comando “**npm i**”.

### Nota

Para validar la versión de loopback instalada en la raíz del proyecto.

#### npm list @loopback/core

Para actualizar versión

#### npm i -g @loopback/cli to update

Actualizar dependencias del proyecto

#### npm update

## Loopback con HTTPS

Instalar en la raíz del proyecto

#### npm install https fs

* Configurar Index.ts que se encuentra en src

import \* as path from 'path';

import { ApplicationConfig } from '@loopback/core';

import \* as https from 'https';

import \* as fs from 'fs';

import { IssBackendApplication } from './application';

export { IssBackendApplication } from './application';

// Configuración básica del servidor Rest

export *const* config: ApplicationConfig = {

  rest: {

    port: +(process.env.PORT ?? 3001),

    host: process.env.HOST ?? 'localhost',

    listenOnStart: false,  // Evita que LoopBack inicie el servidor automáticamente

  },

};

if (require.main === module) {

  (async () *=>* {

*const* app = new IssBackendApplication(config);

    // Inicializa y arranca la aplicación

    await app.boot();

    // Rutas de certificados SSL dependiendo del sistema operativo

*const* sslKeyPath = process.platform === 'win32'

      ? 'C:\\certificados\\nginx-selfsigned.key'  // Reemplaza esta ruta

      : '/etc/nginx/ssl/nginx-selfsigned.key';

*const* sslCertPath = process.platform === 'win32'

      ? 'C:\\certificados\\nginx-selfsigned.crt'  // Reemplaza esta ruta

      : '/etc/nginx/ssl/nginx-selfsigned.crt';

    // Carga los certificados SSL

*const* options = {

      key: fs.readFileSync(sslKeyPath),

      cert: fs.readFileSync(sslCertPath),

    };

    // Crea y arranca el servidor HTTPS usando el manejador de solicitudes de LoopBack

*const* server = https.createServer(options, app.requestHandler);

    server.listen(config.rest.port, () *=>* {

      console.log(`LoopBack is now running on HTTPS at https://localhost:${config.rest.port}`);

    });

  })().catch(*err* *=>* {

    console.error('Error starting the application:', *err*);

    process.exit(1);

  });

}

* Configurar Aplication.ts que se encuentra en src

import {BootMixin} from '@loopback/boot';

import {ApplicationConfig} from '@loopback/core';

import {

  RestExplorerBindings,

  RestExplorerComponent,

} from '@loopback/rest-explorer';

import {RepositoryMixin} from '@loopback/repository';

import {RestApplication} from '@loopback/rest';

import {ServiceMixin} from '@loopback/service-proxy';

import path from 'path';

import {MySequence} from './sequence';

import { StrategyAdmin } from './strategies/admin.strategy';

import { AuthenticationComponent, registerAuthenticationStrategy } from '@loopback/authentication';

export {ApplicationConfig};

export *class* IssBackendApplication extends BootMixin(

  ServiceMixin(RepositoryMixin(RestApplication)),

) {

  [*x*: *string*]: *any*;

*constructor*(*options*: ApplicationConfig = {}) {

    super(*options*);

    // Set up the custom sequence

    this.sequence(MySequence);

    // Set up default home page

    this.static('/', path.join(\_\_dirname, '../public'));

    // Customize @loopback/rest-explorer configuration here

    this.configure(RestExplorerBindings.COMPONENT).to({

      path: '/explorer',

    });

    this.component(RestExplorerComponent);

    this.projectRoot = \_\_dirname;

    // Customize @loopback/boot Booter Conventions here

    this.bootOptions = {

      controllers: {

        // Customize ControllerBooter Conventions here

        dirs: ['controllers'],

        extensions: ['.controller.js'],

        nested: true,

      },

    };

    //registrar estrategia

    registerAuthenticationStrategy(this, StrategyAdmin);

    this.component(AuthenticationComponent);

  }

}

## Crear usuario en MongoDB

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente Texto

Descripción generada automáticamente con confianza media

mongodb+srv://mundochic:o1pARquc62CRsv3i@clbr001.seq4mgj.mongodb.net/mundochic\_db

## Creación de datasource

Para la creación de un datasource se debe seguir los siguientes pasos:

Texto

Descripción generada automáticamente

* Data source debe quedar de la siguiente manera

import {inject, lifeCycleObserver, LifeCycleObserver} from '@loopback/core';

import {juggler} from '@loopback/repository';

*let* config = {

  name: 'mongodb',

  connector: 'mongodb',

  //url: 'mongodb+srv://mundochic:o1pARquc62CRsv3i@clbr001.seq4mgj.mongodb.net/mundochic\_db',

  url: 'mongodb://mundochicuser:YXn32%23XWj%25za0ws%24@192.168.0.19:27017/mundochicdb',

  //url: 'mongodb://mundochic:o1pARquc62CRsv3i@192.168.0.13:27017/mundochic\_db',

  //host: 'clbr001.seq4mgj.mongodb.net',

  //host: '192.168.0.7',

  //port: 27017,

  //user: 'mundochic',

  //password: 'o1pARquc62CRsv3i',

  //database: 'mundochic\_db',

  useNewUrlParser: true

};

// Observe application's life cycle to disconnect the datasource when

// application is stopped. This allows the application to be shut down

// gracefully. The `stop()` method is inherited from `juggler.DataSource`.

// Learn more at https://loopback.io/doc/en/lb4/Life-cycle.html

@lifeCycleObserver('datasource')

export *class* MongodbDataSource extends juggler.DataSource

  implements LifeCycleObserver {

  static dataSourceName = 'mongodb';

  static readonly defaultConfig = config;

*constructor*(

    @inject('datasources.config.mongodb', {optional: true})

*dsConfig*: *object* = config,

  ) {

    super(*dsConfig*);

  }

}

## Creación de modelos

Para la creación de un modelo se debe seguir los siguientes pasos:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## Creación de repositorios

Para la creación de un repositorio se debe seguir los siguientes pasos:

Texto

Descripción generada automáticamente

## Creación de controladores

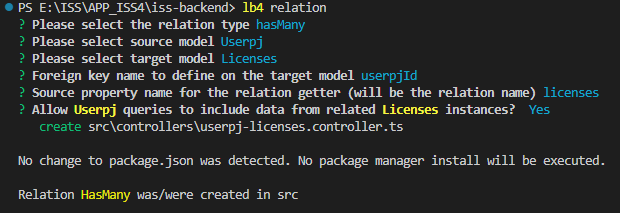
Para la creación de un controlador se debe seguir los siguientes pasos:

Texto

Descripción generada automáticamente

## Relaciones en MongoDB – hasMany

Un cliente puede tener una o más licencias, se utiliza “**hasMany**” relación de uno a muchos.



### Nota

Se crea foránea automáticamente en el modelo de “**licenses**”.

Texto

Descripción generada automáticamente

Relación queda creada en modelo de “**userpj**”.

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

## Relaciones en MongoDB – belongsTo

Una licencia solo puede pertenecer a un cliente, se utiliza “**belongsTo**” relación uno a uno.

Texto

Descripción generada automáticamente

### Nota

Relación queda creada en el modelo de “**licenses**”.



## Relaciones en MongoDB – Uno a uno

Un usuario tiene un login y el login pertenece a un solo usuario. (**belongsTo donde irá la ForeingKey**).

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## Creando usuario login automático luego de registrarse

* En el controlador de userpj, debemos importar el repositorio de login y en el constructor debemos inyectarlo para acceder a las funcionalidades del **CRUD** propias del login.

Texto

Descripción generada automáticamente

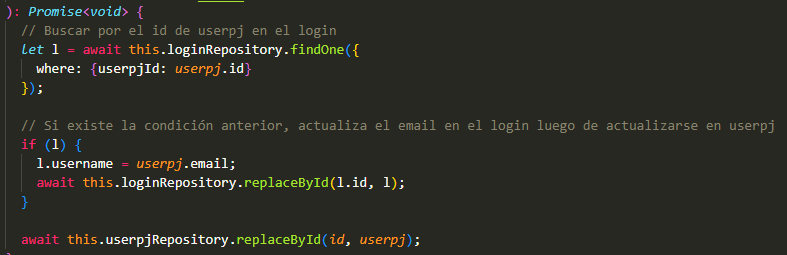
* En el método “**post**” se debe asociar un login a un userpj ya creado. Se pone “**await**” para que este espere una respuesta y luego de recibirla siga con el código y no siga derecho.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

## Actualizar login al actualizar el usuario

En el método “**put**” del controlador de userpj se debe buscar el login que está asociado al usuario que se está actualizando.



## Servicios Encrypt – Decrypt

Crear el servicio de la siguiente manera:

Texto

Descripción generada automáticamente

* Instalación de librerías





* Instalación de dependencias para la librería de crypto



* Importar librería en el servicio creado



* Crear métodos de encriptación y desencriptación

Texto

Descripción generada automáticamente

* Importamos el servicio de **Encrypt – Decrypt** anteriormente creado y la librería de **service** en el controlador donde vamos a utilizarlo, en este caso en el **userpj-licenses.controller.ts**

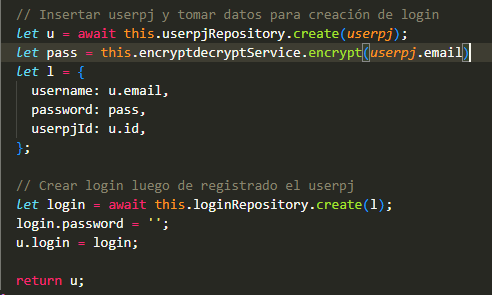


* Inyectamos el servicio de **Encrypt – Decrypt** en el constructor para poder utilizarlo.

Texto

Descripción generada automáticamente

* En el “**post**” y en la parte del “**promise**” hacemos el llamado del servicio de encriptación.



## Servicio – Autenticación

* Crear servicio de autenticación.

Texto

Descripción generada automáticamente

* En el anterior servicio creamos el método para la autenticación del usuario.

Texto

Descripción generada automáticamente

### Nota

Tener en cuenta las librerías y las inyecciones.

Texto

Descripción generada automáticamente

* Se debe crear un modelo tipo **model**, esto con **lb4 model**, en este caso se llamó “**userlogin”** con los campos que enviamos como parámetro que son **email** y **password** para compararlos con el modelo **login** y me genere el token.

Texto

Descripción generada automáticamente

* En el controlador de **userpj** se debe crear un método **post/login** que consumirá el servicio anteriormente creado.

Texto

Descripción generada automáticamente

## Token

* Instalar librería JWT

Texto

Descripción generada automáticamente

* En el servicio “**token**” se importa la librería, en caso de salir error se debe instalar los “**types**” del **JWT**.



Texto

Descripción generada automáticamente

* Crear servicio llamado “**token**” y dentro de este crear métodos de generación y validación del token.

Texto

Descripción generada automáticamente

### Nota

Tener en cuenta las librerías a importar.

Texto

Descripción generada automáticamente

* En el controlador de **userpj** se debe ajustar el **post** “**login**” para consumir el servicio de generar token creado anteriormente.

Texto

Descripción generada automáticamente

## Autorización

* Instalar librerías para la autenticación.



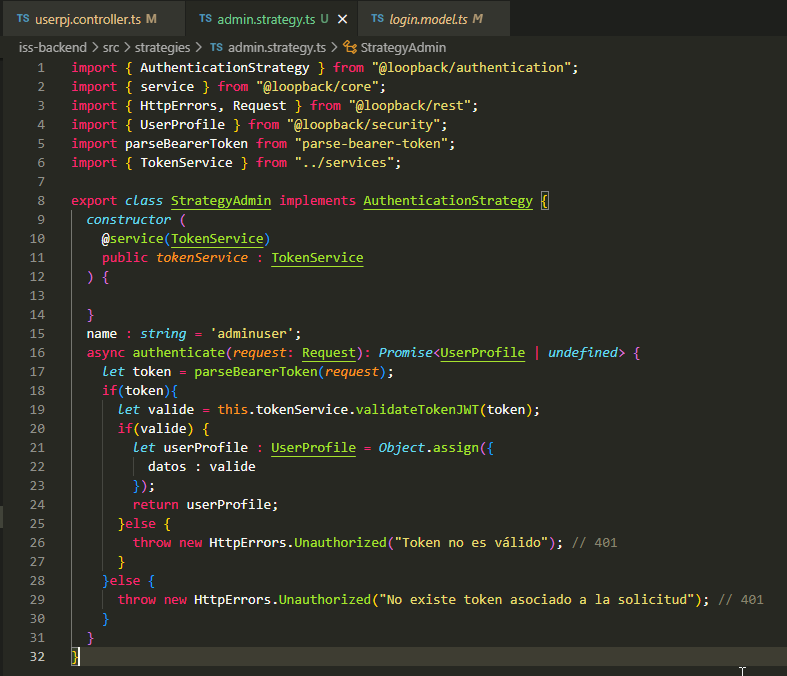




* Crear carpeta y archivo para crear estrategias para el administrador dentro de “**src**”.



* Creamos estrategia para el rol “**admin**”



* Registrar la estrategia en “**application.ts**” dentro de **src** para que esta funcione.

Texto

Descripción generada automáticamente

### Nota

Tener en cuenta las librerías a importar



* En el controlador se debe implementar la estrategia creada en el método que queramos bloquear, en este caso en la consulta de los usuarios. (bloquea un solo método).

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

* Para bloquear todos los métodos en general se implementa la estrategia antes de la clase del controlador.

Texto

Descripción generada automáticamente

* Para generar excepciones de métodos.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## Cerrar sesión

* En el archivo ts del menú creamos un método y está la dirigimos a la página que queramos.

Texto

Descripción generada automáticamente

* En el archivo ts de **exit** ejecutamos el método salir creado en el **security.service.ts** que se encuentra en los servicios de authentication y redireccionamos hacia el index, esto se hace en el **OnInit** “se ejecuta apenas carga la URL”.

Texto

Descripción generada automáticamente

## Guardian

* Creamos el guardián de la siguiente forma:

Texto

Descripción generada automáticamente

* En el guardián creado anteriormente, se debe configurar de la siguiente forma:

Texto

Descripción generada automáticamente

### Nota

Tener en cuenta los métodos que se utilizan desde **token.service.ts**

Texto

Descripción generada automáticamente

* En el routing del módulo se debe agregar el **canActivate** sobre cada ruta

Texto, Chat o mensaje de texto

Descripción generada automáticamente