The background of the entire page is a close-up, artistic photograph of a film reel. A large, curved section of the reel is visible on the left side, showing the dark film strip as it winds around the metal hub. The lighting is soft, creating a sense of depth and texture. In the bottom right corner, a straight section of the film strip is visible, showing the characteristic sprocket holes. The overall color palette is muted, with greys, blacks, and soft whites.

realizado por

ALICIA
CARRASCOSO LOZANO

INDECISIONCINEMA

PROYECTO FINAL DE GRADO DAIM
PARA IES FRANCISCO DE GOYA

Índice

1.	Introducción	5
1.1	Objetivos iniciales.....	5
1.2	Tecnologías	6
1.2.1	CSS3	6
1.2.2	JSON	6
1.2.3	JavaScript.....	6
1.2.4	MySQL y MySQL Workbench.....	6
1.2.5	Visual Studio Code.....	6
1.2.6	GitHub	7
1.2.7	GitKraken.....	8
1.2.8	Node.js y Librerías NPM	9
1.2.8.1	Pug	9
1.2.8.2	Bootstrap.....	10
1.2.8.3	Express	11
1.2.8.4	Debug.....	11
1.2.8.5	HTTP-Errors	11
1.2.8.6	jQuery	11
1.2.8.7	md5	12
1.2.8.8	Morgan.....	12
1.2.8.9	Popper.....	12
1.2.8.10	Serve-Favicon.....	13
1.2.8.11	Multer.....	13
1.2.9	Dependencias: Nodemon	13
2.	Análisis de requisitos	14
2.1	Homepage.....	14
2.2	Cartelera	15
2.3	Formulario Login.....	16
2.4	Formulario Registro	17
2.5	Panel Administración	18
2.5.1	Control de la pestaña Admin.....	19
2.6	Sala Cine.....	20
2.7	Resume	20
3.	Adecuación del entorno de trabajo	22

3.1 Visual Studio Code.....	22
3.2 NodeJS & Express & NPM.....	22
3.2.1 Express.....	24
3.2.2 Instalación librerías NPM	24
3.3 MySQL y MySQL Workbench	25
4. Arquitectura de Proyecto	26
4.1 Estructura y Diseño	26
4.1.1 Bin.....	26
4.1.2 database	27
4.1.3 node_modules.....	27
4.1.4 public	28
4.1.5 routes	28
4.1.6 views.....	29
4.1.7 Resto de files	29
5. Desarrollo del Proyecto.....	30
5.1 Conexión a BBDD.....	30
5.2 Login y creación de la Sesión.....	30
5.3 Logout y destrucción de la Sesión	32
5.4 Registro de Usuario	32
5.5 Cartelera	33
5.6 Funciones Admin	34
5.6.1 Creación y Borrado de Salas.....	34
5.6.2 Creación y Borrado de Películas	35
5.6.3 Selección y actualización de asientos.....	37
6. Desarrollo de la BBDD	39
6.1 Diagrama Entidad Relación.....	39
6.2 Script.....	40
6.3 Tablas	42
6.3.1 Usuario	42
6.3.2 Película	43
6.3.3 Sala	43
6.3.4 Asientos.....	44
6.3.5 Tabla Entrada	44
7. Pruebas.....	45
8. Conclusión.....	47
8.1 Dificultades.....	47

8.2 Alcance y Limitaciones.....	47
8.3 Líneas futuras de investigación	48
8.4 Conclusión.....	49

1. Introducción





El documento presente es la memoria del Proyecto de Fin de Ciclo Formativo de DAIM (Desarrollo Aplicaciones Informáticas Multiplataforma)

Indecision Cinema es una aplicación web alojada en servidor, en la que se gestiona un cine.

1.1 Objetivos iniciales

Como finalidad, el proyecto pretende construir una página web el lenguaje de programación Nodejs, usando una temática basada en un cine y la gestión del mismo.

Los objetivos iniciales que se querían abarcar y superar son:

-  Ampliación de los conocimientos en lenguajes de programación Open Source (Node en este caso)
-  Aprendizaje y familiarización con diferentes herramientas para el control de proyectos y entornos de desarrollo (IDE)
-  Uso y control del modelo a tres capas Model View Presenter (MVP)
-  Creación de una Aplicación Web atractiva para el usuario, rápida en petición y respuesta y eficiente en el ejemplo para comprobar que se puede aplicar a cualquier aplicación que tenga una finalidad similar.

1.2 Tecnologías

Los lenguajes de programación/marcas y tecnologías/entornos usados son:

1.2.1 CSS3



CSS (*Cascading Style Sheets*) u “*Hojas de Estilo en Cascada*”, es un lenguaje de diseño gráfico para la presentación visual de un documento web e interfaces como HTML o XHTML. En el caso de este proyecto, se ha usado para poder dar estilo junto con el lenguaje de marcas Pug.

La forma correcta de importarlo sería desde el Header del .pug:

```
link(rel='stylesheet', href='/stylesheets/mystyle.css')
```

1.2.2 JSON



Es el acrónimo de *JavaScript Object Notation*. Con un formato de texto se ha usado en el proyecto para el intercambio de datos, ya sea para mensajes de cara al usuario o procesos entre capas de la BBDD y Presentación.

1.2.3 JavaScript



Es un lenguaje de programación interpretado y orientado a objetos. En el caso de este proyecto, se usará junto con Nodejs y sus librerías para crear páginas dinámicas corriendo en servidor web.

1.2.4 MySQL y MySQL Workbench



Es un sistema de gestión de BBDD relacional de código abierto. Junto con MySQL Workbench se usará en el proyecto para la creación y gestión de la base de datos.

1.2.5 Visual Studio Code

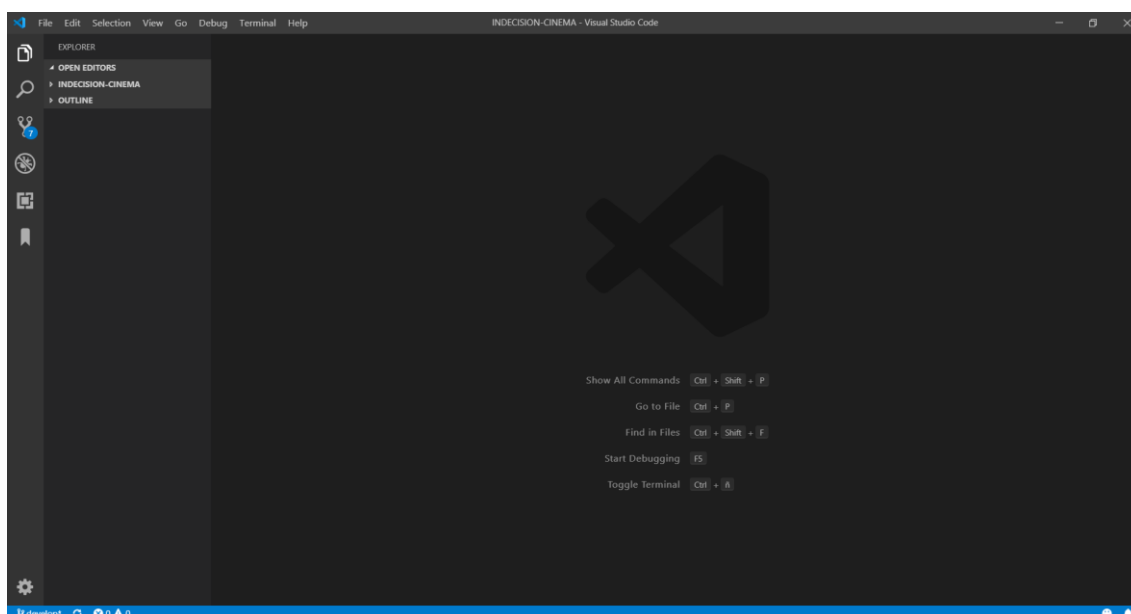


Es el editor escogido para llevar a cabo el proyecto. Es un editor de código fuente desarrollado para Microsoft para Windows, Linux y macOS. Es gratuito y de código abierto. Contiene muchos plugins útiles para la depuración y resaltado del código.

Su estructura es limpia y ligera a la vista, con el panel de proyectos a la izquierda y en la cabecera las diferentes opciones para configurar.

Además, contiene funcionalidades ya incluidas como el: Resaltado de sintaxis, autocompletado de código, refactorización y opción de visualizar una consola que podemos personalizar según el SO que queramos y podemos hacer debug en ella.

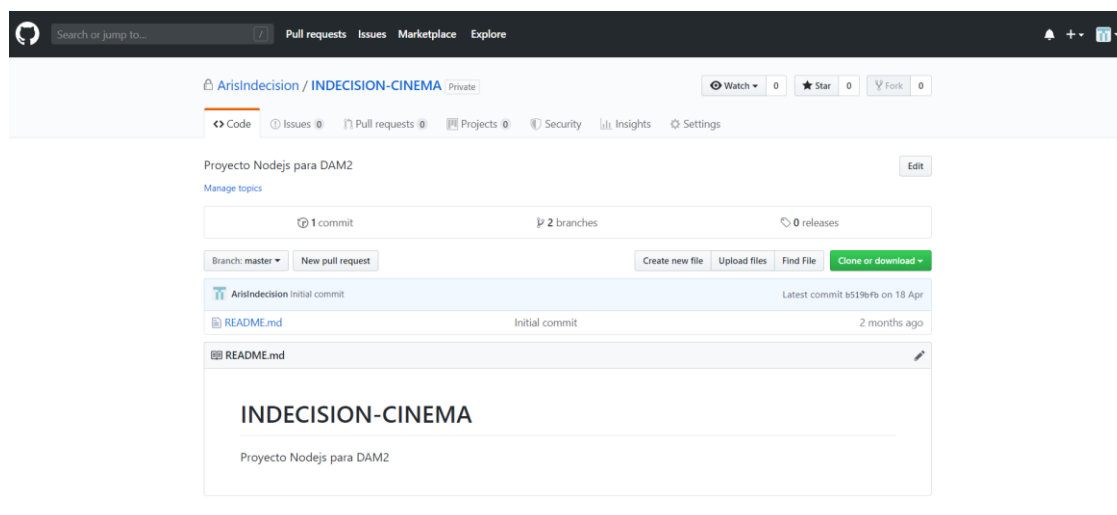
A la hora de abrir cualquier proyecto, podemos hacerlo de forma sencilla cogiendo la carpeta contenedora y arrastrándola directamente a la aplicación. De esta forma se nos añadirá todo el directorio con las diferentes folders y files que tengamos en un segundo para empezar a editar nuestro código de una mucho más cómoda.



1.2.6 GitHub



Es una plataforma que se usa para alojar proyectos utilizando un sistema de control de versiones Git. El uso que se le da en este proyecto es sencillamente el poder guardar y tener en un repositorio local y remoto el código para poder modificarlo y usarlo en cualquier máquina a la que se acceda y descargue el código del repositorio.

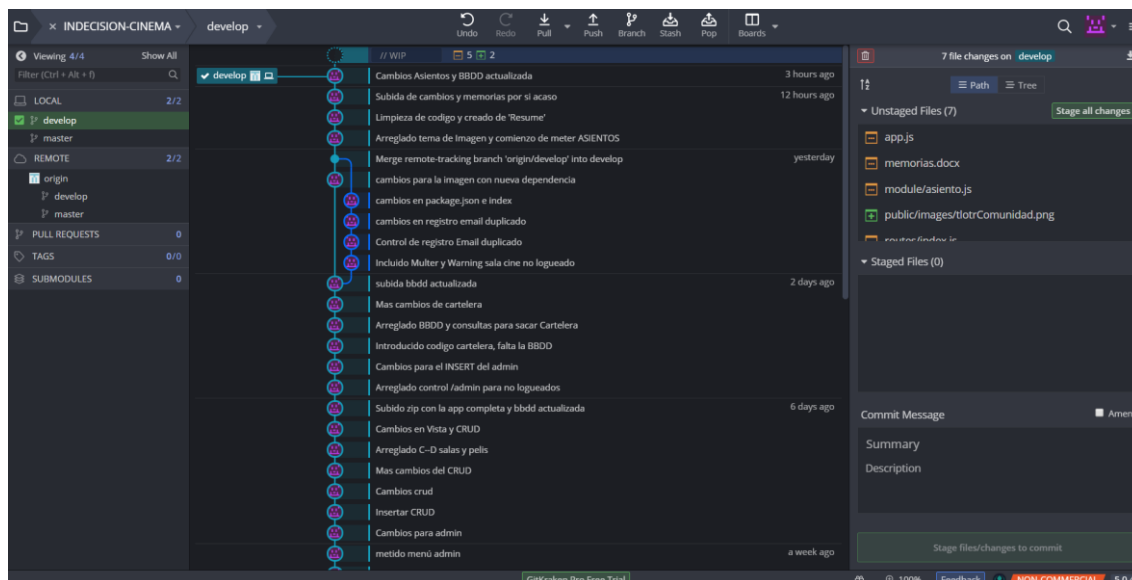


En mi proyecto tanto el GitHub como el GitKraken que explicaré a continuación están íntimamente relacionados, ya que ambos se pueden alinear para trabajar los repositorios en local y guardarlos en remoto con un clic y de forma visual que ayuda al usuario.

1.2.7 GitKraken

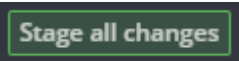


Es un programa de gestión de Git con un diseño ramificado para tener una visualización del trabajo del equipo. En este caso, aunque únicamente lo usaba yo, lo tenía como control de commit ya que se pueden dejar comentarios de cada cambio que se sube a repositorio. En esta aplicación también se pueden gestionar, clonar y abrir diferentes repositorios para tener todos organizados en una misma ventana a golpe de click.

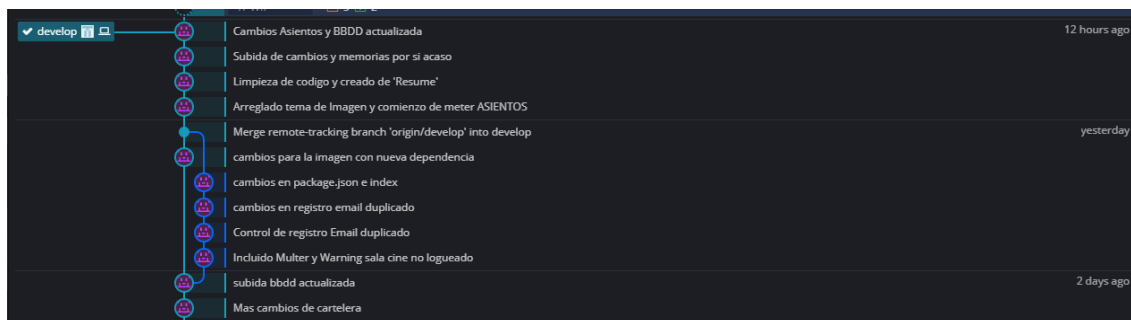


Comenzando desde arriba izquierda encontramos el nombre del repositorio y la rama en la que se está trabajando. Tenemos justo al lado siguiendo en la cabecera las opciones de *undo* (para volver atrás en alguna acción que hayamos hecho), *redo* (volver hacia adelante en la acción), *pull* (para bajarnos los cambios de la rama que nos interese del repositorio en remoto a nuestro local debidamente actualizado), *push* (al pinchar sobre esa opción subiremos los cambios de la rama en la que nos encontremos a la rama del mismo nombre en remoto tras haber realizado un commit), *Branch* (desde esta opción, habiéndonos situado en la rama directorio que queramos, podremos crear y nombrar una rama personalizada para nuestro uso o para el uso que queramos darle), *Stash* (es un botón de cambios en caliente, que nos hace "fotos" previos a los commit para luego subirlos a remoto si finalmente efectuamos el commit), *Pop* (nunca suele usarse esta opción, pero está para deshacer el *stash*).

En la parte de la derecha se nos muestran a tiempo real los cambios que hemos hecho en el proyecto, con las opciones de borrado esos "*Stages*" para deshacer los cambios directamente en un solo clic, o si por el contrario queremos committear, daríamos al

botón  donde podremos añadir un comentario sobre lo que vamos a guardar con el commit para a la hora de subirlo con un *push*, aparezca en el resumen ramificado y los compañeros sepan de qué trata con solo un vistazo.

Por último tenemos la zona central donde se encuentran las ramas de forma visualmente presentadas.



A la izquierda nos aparecerá con el nombre de la misma y los commits que se han efectuado. El logo que aparece al lado de la descripción corresponde a la persona que ha efectuado dicho *commit* y *push* al repositorio remoto. Por último a la derecha podemos encontrar la fecha o el momento en el que se ha efectuado dicha acción.

Esta herramienta de gestión de proyectos Git es muy útil, ya que al hacer un *commit* y *push* subimos los cambios a remoto, pero siempre podremos volver a versiones anteriores si pinchamos en el momento de la ramificación que nosotros elijamos.

Además, podremos hacer *merge* entre ramas de forma sencilla, cerciorándonos de que el código de todos los compañeros que estén trabajando en el proyecto esté siempre actualizado a la última versión.

1.2.8 Node.js y Librerías NPM



Nodejs es un entorno en tiempo de ejecución (*runtime*) de JavaScript para servidor multiplataforma, de código abierto y enfocado a la capa de servidor, con una arquitectura orientada a eventos y corriendo en el motor V8 de Google. Todo el código se ejecuta en el lado del servidor. Y Tiene una amplia gama de librerías open source como las que se muestran a continuación, que tienen sus propias funciones individuales.

NPM es el administrador de paquetes predeterminado de Nodejs. El administrador de paquetes permite a los programadores publicar y compartir más fácilmente el código fuente de las bibliotecas de Nodejs y está pensado para simplificar la instalación, la actualización y desinstalación de las mismas.

1.2.8.1 Pug



Es un motor de plantillas de alto rendimiento, implementado con JavaScript para Nodejs y navegadores. Pug es el reinventado Jade. En este proyecto se usará para crear las plantillas de estilo que luego con CSS dará forma a las páginas para la vista del usuario. La forma de escribirse, es similar al HTML pero sin las marcas de las etiquetas `<` `>` e indentado. Además, se le puede añadir lógica dentro de la propia página de pug para controles más específicos.

A continuación se muestra un ejemplo de cómo se transformaría un código HTML convencional.

```
.content
```

```
<div class="content"></div>
```

```
#content
```

```
<div id="content"></div>
```

Además, también se puede añadir lógica en los .pug y comentarios.

```
- for (var x = 0; x < 3; x++)
  li item
```

```
<li>item</li>
<li>item</li>
<li>item</li>
```

```
if (session.admin === 0)
  li
    a.nav-link(href="admin") Admin
if (session.loggedin)
  li
    a.nav-link(href="logout") Logout
else
  li
    a.nav-link(href="login") Login
```

Y scripts con código.

```
script.
  let id_asientos = [];

  $(".libre").on('click',function(event) {
    console.log("paso por libre a seleccionado");
    // Aquí lo que hago es: de aque que produce el evento, que el 'click', regoco el taget sobre el que se ha producido, es decir,
    // la butaca sobre la que he clicado y de ella recogo el id. Y así sé la butaca elegida.
    // Vale, lo he cambiado a $(this), porque si buscas en google, es el sinónimo de event.currentTarget y así, puedes decir que usar
    // y es más corto y sencillo. Que diver programar peo.
    var id_butaca = event.currentTarget.id;
    id_asientos.push(id_butaca);
    $(this).removeClass("libre");
    $(this).addClass( "seleccionado" );
  });
```

1.2.8.2 Bootstrap



Es una biblioteca multiplataforma de código abierto para diseño de sitios y aplicaciones web. Se usa para diseño basado en HTML, CSS y JavaScript. En este proyecto se usará junto con Pug y CSS3 para dar estilo al front-end de las páginas, como por ejemplo la barra de navegación o los formularios de registro y login.

Se instalará con el comando por consola: `npm install bootstrap`, de esta forma se añadirá directamente a nuestra carpeta de node_modules del proyecto. Una vez creado, tendremos que importarlo a nuestro file .pug

```
script(src='/bootstrap/js/bootstrap.js')
```

Y además incluir la línea de código en Node donde lo usaremos indicaremos la ruta de donde debemos coger la librería para poder usarla dándole un alias o ruta acortada para que nos sea más cómodo:

```
app.use('/bootstrap', express.static(__dirname +
  '/node_modules/bootstrap/dist')); // use bootstrap
```

1.2.8.3 Express

ex Express es un framework para Nodejs que sirve para ayudar a crear aplicaciones web en menos tiempo, ya que nos proporciona funcionalidades para gestionar diferentes utilidades como sesiones y cookies.

De Express también se usará la extensión Flash (*Express-Flash*), que son mensajes definidos para procesar y redirigir solicitudes. En este proyecto se usará para dar al usuario información concreta con ayuda de los mensajes.

Además, se implementará la extensión Session (*Express-Session*), con lo que podremos crear un middleware de sesión con las opciones dadas. Los datos de sesión no se guardarán en una cookie, únicamente nos quedaremos con el ID de sesión, ya que los datos se almacenan en el lado del servidor.

Otra extensión o framework que también se usará de express es *Cookie-Parser*, es un recurso muy común para el tratamiento de cookies.

1.2.8.4 Debug

Es una librería NPM para Nodejs que se encarga de depurar y exponer una función, y ésta devolverá una versión decorada por *console.log()* para que sea más agradable y sencillo a la vista.

1.2.8.5 HTTP-Errors

Librería NPM para Nodejs que ayuda a crear funciones de errores predefinidos para directamente mostrar aquel que mejor nos convenga.

Para usarlo correctamente tendremos que crearnos una variable en el proyecto que guarde la librería de forma requerida para poder hacer uso de ella:

```
let createError = require('http-errors');
```

1.2.8.6 jQuery

jQuery en el proyecto se incluye directamente como dependencia de Nodejs.

En su definición, se trata de una biblioteca multiplataforma de JavaScript que permite simplificar la manera de interactuar con los documentos HTML (Pug en mi caso).

La forma de instalarlo como cualquier librería de NPM de Node, es realizando el comando por consola:

```
npm install jquery
```

Y después importado en el Header de nuestro file de .pug para poder usar la librería correctamente.

```
script(src='/jquery/jquery.js')
```

Además, en el código habrá que introducir una línea nueva, en la que requiramos de jQuery para que funcione y no nos salte un error por consola:

```
app.use('/jquery', express.static(__dirname +  
'/node_modules/jquery/dist')); // use jquery
```

Inicaremos de esta forma de dónde tiene que coger jQuery y cómo queremos llamarlo.

1.2.8.7 md5

Librería o paquete NPM para Nodejs que se puede usar para el lado del servidor y el lado del cliente. Se usa para poder encriptar o cifrar en md5. En el caso de este proyecto, se usa para registrar la Password del usuario registrado en BBDD de forma cifrada para mayor control y seguridad.

1.2.8.8 Morgan

Morgan es un middleware de registro de solicitudes HTTP para Nodejs. Simplifica así el proceso de registro de solicitudes para la aplicación. El formato de una función con Morgan se llamará con tres argumentos: tokens, req y res, donde los tokens son un objeto con todos los tokens definidos, req es la solicitud HTTP y res es la respuesta HTTP. Por norma general, se espera que la función devuelva una cadena que será la línea de registro, o por el contrario undefined o null para omitir el registro.

1.2.8.9 Popper

Popper es una librería de Nodejs que se implementa en el HTML (Pug en este caso) como un script de JavaScript para poder usar sus funcionalidades y ventajas. Finalmente no se usa en este proyecto con todo su potencial, ya que con Pug, CSS, Bootstrap y JavaScript se consiguen todas las funcionalidades deseadas por el momento.

Como librería de Nodejs NPM tendremos que usar el comando: `npm install popper` para poder usarlo. Lo llamaremos desde el .pug como un script importado:

```
script(src='/popper/popper.js')
```

Y añadiremos la línea de código necesaria en Node para poder usarlo, indicando la ruta donde se encuentra en las librerías de node_modules:

```
app.use('/popper', express.static(__dirname +  
'/node_modules/popper.js/dist')); // use popper
```

1.2.8.10 Serve-Favicon

Es un middleware de Nodejs para poder mostrar en la página un favicon. El módulo almacena en caché el icono en memoria para mejorar el rendimiento.

1.2.8.11 Multer

Multer es un middleware de Nodejs para manejar subida de archivos (uno o multi) para mayor eficiencia los archivos subidos a proyecto o BBDD son guardados en formato binario, por lo que luego para poder usar por ejemplo imágenes en el proyecto se ha tenido que usar métodos para convertir desde binario a formato PNG o JPG.

```
/**
 * Uso de MULTER para la subida de imagen a proyecto con formato y nombre original
 */

// middleware MULTER Nombre original
let storage = multer.diskStorage({
  destination: path.join(__dirname, 'public/images'),
  filename: (req, file, cb) => {
    cb(null, file.originalname);
  }
});

// route MULTER
app.use(multer({
  storage: storage,
  dest: path.join(__dirname, 'public/images')
}).single('imagen'));

app.post('/subirImagen', (req, res) => {
  //console.log(req.file);
  req.flash('success_messages', 'Se ha añadido correctamente.');
```

```
res.redirect('/admin');
});
```

1.2.9 Dependencias: Nodemon












En las dependencias de Nodejs podemos encontrar una herramienta de lo más útil como es Nodemon.

Nodemon es una herramienta que ayuda a desarrollar aplicaciones basadas en Nodejs. Su trabajo consiste en reiniciar automáticamente la aplicación tras guardar cambios en el archivo directorio del proyecto. Nodemon no requiere cambios adicionales en su código o método de desarrollo.

Por ello en este proyecto ha sido de suma importancia para ahorrar tiempo a la hora de ejecutar cambios y ver los resultados de forma inmediata tras su guardado, sin necesidad de tener que arrancar de nuevo el servidor o aplicación.

2. Análisis de requisitos

El proyecto tiene como motivo la creación de un Cine con sus características, a diferenciar entre:

-  Un frame Index o Home donde poder tener una presentación y primera vista de la aplicación.
-  Es necesario tener un menú donde se encuentren todas las opciones que el usuario ya sea admin o no, pueda utilizar.
-  Un login donde los usuarios puedan loguearse en la aplicación. Gestionando los posibles errores de usuario o password con el aviso correspondiente.
-  Un formulario de registro a la aplicación para nuevos usuarios. Gestionando los posibles errores que puedan darse como el duplicado de email.
-  Una página que sea el muestrario de las películas que el usuario puede seleccionar para poder ver.
-  Una página ligada a la cartelera anterior, que muestre los asientos de la sala de cine para que el usuario vea los ocupados y libres y así poder escoger los que prefiera para su gestión.
-  Un resumen de la elección del usuario de forma detallada bien visualizada claramente en pantalla.
-  Una página de Administración donde se controla que únicamente el usuario admin pueda acceder. En ella se encuentran las funcionalidades de creación y borrado de carteleras y películas para publicar en la página de acceso al usuario.
-  Y por último y no menos importante, la opción de deslogueo.

A continuación se presentan los análisis de requisitos y funcionalidades de las diferentes páginas que componen el proyecto:

2.1 Homepage

La funcionalidad de la página principal/Home o Homepage es informar al usuario en una vista rápida el contenido que puede encontrar en la misma. Aun así, no se encuentran todas las funcionalidades, ya que para poder acceder a todas, se requiere de estar logueado, ya sea un usuario al uso o un administrador.

En la página se puede ver un navegador con las diferentes opciones según si estas logueado (admin/user) o deslogueado.

Deslogueado:



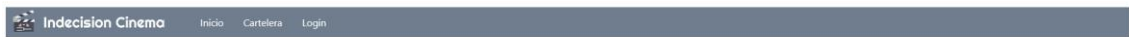
Logueado User: donde se aprecia que la opción de 'Login' desaparece y en su lugar encontramos la opción de deslogueo 'Logout'.



Logueado Admin: y aquí se puede ver que aparte de la opción de 'Logout' aparece también la opción que nos llevaría al apartado del panel administrativo: 'Admin'.



Inmediatamente debajo de la barra de navegación encontramos una sección de las películas que saldrán próximamente. Es estático y con funcionalidad meramente decorativa. Además, son enlaces que nos llevan a una página donde encontramos más detalles de la película seleccionada.



Bienvenido a Indecision Cinema

Próximos estrenos



2.2 Cartelera

La Cartelera es una página que está visible para todo usuario, ya sea *user* normal, *admin* o incluso si se está deslogueado.

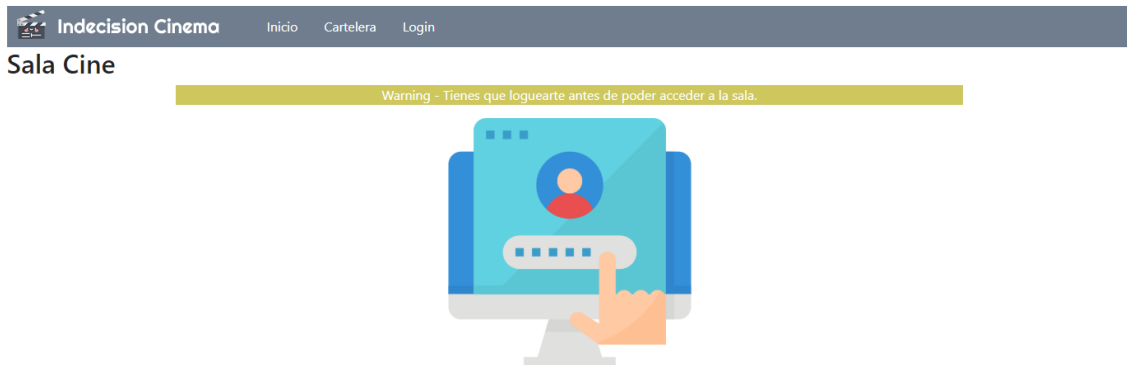
En ella encontraremos las películas ofertadas con todos los datos. Desde ahí, pinchando sobre la película deseada se podrá acceder a la sala correspondiente y comprar nuestros asientos.



Cartelera



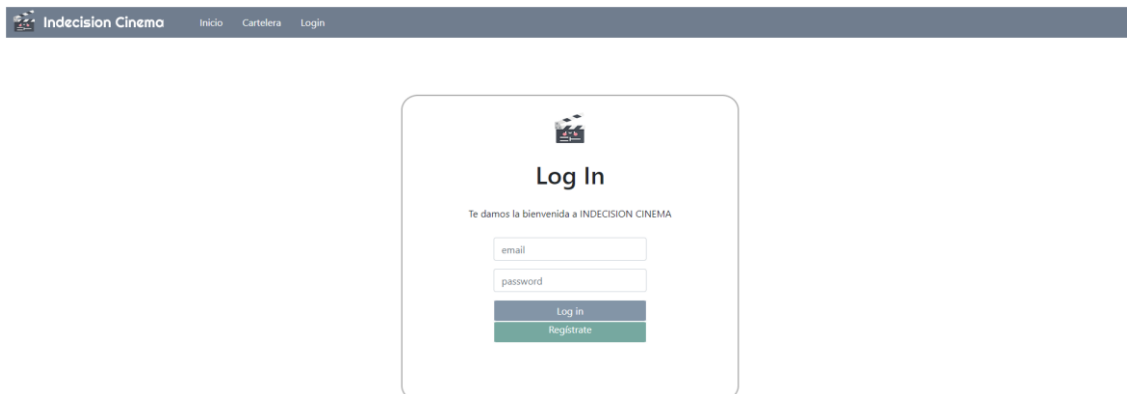
Una vez se está en la cartelera, se deberá estar logueado para poder comprar los asientos, por el contrario no podrá ser redirigido a la página de compra de asientos y se le conducirá a una pantalla de aviso o *warning*.



2.3 Formulario Login

Como comentamos más arriba, para poder acceder a todas las funcionalidades que la página nos puede ofrecer, tenemos que estar logueados previamente. En el caso de no estar logueados, solo tendremos acceso a la página principal (Inicio en el navegador) y a la Cartelera para informarnos de las películas que se ofertan.

Dentro del Login, se encuentra la opción de '*Registrarse*' en el caso de que el usuario que quiera entrar a la página no tenga una cuenta previa en la aplicación.



El acceso a la página será con el email y la contraseña, los cuales son requeridos.

En el caso de que el Login no se haya efectuado correctamente, saldrá un mensaje de error avisando al usuario:

2.4 Formulario Registro

Obligatoriamente hay que rellenar el formulario de registro para poder loguear posteriormente en la aplicación y acceder a todas las funcionalidades. Se pedirán datos como: email, nombre, apellidos, DNI y una contraseña. Todos ellos requeridos de forma obligatoria para poder realizar el registro.

En el caso de que todos los datos se hayan añadido correctamente, se almacenará en un registro en la tabla Usuario de la BBDD y se le notificará mediante un mensaje en la misma página del Registro que la operación ha sido exitosa y recuerda al usuario que se le ha enviado un email con los datos introducidos en la web (no funcional) para poder consultarlo en el caso de olvidarse la contraseña o correo en un futuro.

Indecision Cinema Inicio Cartelera Login

Regístrate

Te damos la bienvenida a INDECISION CINEMA

email

dni

nombre

apellidos

contraseña

Registrar

Se ha registrado correctamente. Encontrarás en tu correo todos los datos.

Si la PK (Primary Key) que es el email ya se encuentra registrado en la BBDD, saldrá un mensaje avisando al usuario para que pruebe a registrarse con otro correo.

Indecision Cinema Inicio Cartelera Login

Regístrate

Te damos la bienvenida a INDECISION CINEMA

email

dni

nombre

apellidos

contraseña

Registrar

Email ya registrado. Prueba con otro :)

2.5 Panel Administración

Una se ha logueado como administrador, aparecerá en el navegador la opción de *Admin*, donde el usuario en cuestión podrá tener acceso a la creación y borrado de Salas, publicadas para el usuario que quiera entrar a comprar entradas para películas disponibles y Películas asociadas o no, a las salas. Aquellas películas que estén asociadas a una sala aparecerán en Cartelera para uso y disfrute de los usuarios.

Dentro del panel de administración se pueden diferenciar los formularios de Creación y de Borrado tanto de Cartelera como de Sala.

Además a última hora, se añadió la subida de imagen de 300x400px para poder adjuntarla debidamente desde el propio proyecto a la carátula de la película que aparecerá en Cartelera.

Sólo puede habilitar una película por sala, ya que el número de sala es único. Al crear la sala se le debe indicar el título de la película a proyectar. El borrado de sala se realiza indicando también el número de sala.

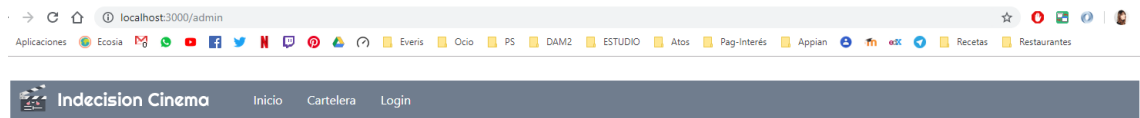
Todo ello quedará registrado en la BBDD en la tabla sala.

La película se crea desde el formulario correspondiente indicando el título de la película, la duración en minutos, una breve descripción y la imagen, la cual se añadirá en el registro y se mostrará en la página Cartelera. El borrado de la película se realizará por el título de la misma.

Todo ello quedará registrado en la BBDD en la tabla película y se avisará con un mensaje al administrador directamente desde la página para que pueda comprobar si la gestión ha finalizado correctamente o por si el contrario ha habido un error.

2.5.1 Control de la pestaña Admin

Si cualquier usuario al uso, o usuario no registrado/logueado que no sea Administrador intenta acceder a la url poniendo en el navegador /admin, se controla que el paso es restringido avisándole que no es una página disponible.

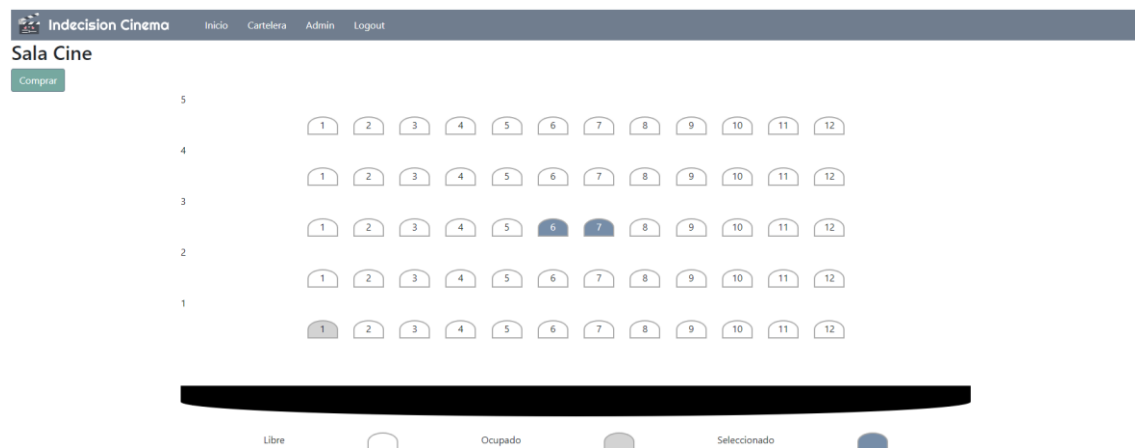


2.6 Sala Cine

Sala de cine será una página a la cual se accede directamente desde la cartelera.

Cuando el usuario accede a la Cartelera estando logueado, puede pinchar sobre la película que desee y ese enlace le llevará directamente a la sala con los asientos en la cual podrá escoger los asientos que quiera para hacer su reserva.

La sala se mostrará de la siguiente manera:




Los asientos en blanco estarán libres, los asientos en gris estarán ocupados y los azules serán aquellos que el usuario haya seleccionado para reservarlos/comprarlos.


2.7 Resume

En la página Resume, mostraremos los detalles de la elección del usuario antes de efectuar la compra de las entradas. En ella se nos mostrará la carátula de la película, el título, número de sala, duración, fecha y hora además de las filas y butacas seleccionadas.

No se ha implementado un botón de "*finalizar compra*" ya que no entraba en el alcance del proyecto.

 Indecision Cinema Inicio Cartelera Logout

Resumen de Compra



Título Película: Avengers Endgame

Número de Sala: 2

Duración en min: 180

Fecha Reproduccion: Fri Jun 14 2019

Hora Reproducción: 21:00:00

Fila: 5

Butaca: 6

Fila: 5

Butaca: 7

Precio: 6,50 Euros/Entrada

3. Adecuación del entorno de trabajo

Las instalaciones para llevar a cabo este proyecto son mínimas, teniendo únicamente que configurar el IDE de trabajo, el IDE de gestión de BBDD y por último la creación del propio proyecto. Esto último gracias a Node & Express se hace de forma automática, generando el proyecto con los folders y files iniciales para que podamos empezar a programar sin demora.

A continuación se presentan las diferentes configuraciones necesarias para poder llevar a cabo todo el proyecto.

3.1 Visual Studio Code

Su instalación es sencilla. Sólo hay que entrar en la página de Visual Studio Code y descargar la versión dependiendo del SO del que dispongamos.

Una vez descargado el ".exe", habrá que ir pulsando "next" hasta finalizar con la instalación. Dentro del propio Visual Studio Code, existen extensiones que podremos implementar para ayudarnos con el código. Pero eso ya es a gusto del consumidor.

3.2 NodeJS & Express & NPM

En esta parte vamos a instalar y configurar toda la parte del servidor de Node. Para ello iremos a la página oficial de Node y descargaremos la versión que sea acorde con nuestro SO. Una vez descargado el instalador, lo iniciaremos y seguiremos los pasos de la instalación haciendo clic en el botón "next".

Para comprobar que efectivamente tenemos Node instalado haremos la prueba desde la consola de Windows de la siguiente manera:

```
node -v
```

```
C:\Users\acarras1\INDECISION-CINEMA\INDECISION-CINEMA>node -v  
v10.15.3
```

Y comprobaremos que tenemos además añadida la versión de NPM:

```
npm -v
```

```
C:\Users\acarras1\INDECISION-CINEMA\INDECISION-CINEMA>npm -v  
6.4.1
```

NPM o Node Package Module es una ayuda para JavaScript para cargar dependencias de forma efectiva. Para poder cargar las dependencias en nuestro proyecto, tendremos que poner:

```
npm install
```

Con este comando, encontrará el fichero package.json y el directorio root de nuestro proyecto e instalará las dependencias que vayamos indicando a medida que el propio "instalador/configurador" nos va requiriendo por pantalla.

Otra forma de iniciar directamente un proyecto sería con el siguiente comando:

```
npm init
```

Y directamente se instalará y comenzará con la creación del proyecto y el package.json mencionado anteriormente.

```

1  {
2    "name": "indecision-cinema",
3    "version": "1.0.0",
4    "description": "Proyecto cine basado en Node",
5    "private": true,
6    "scripts": {
7      "start": "nodemon ./bin/www"
8    },
9    "dependencies": {
10     "bootstrap": "^4.3.1",
11     "cookie-parser": "^1.4.4",
12     "debug": "~2.6.9",
13     "express": "^4.16.0",
14     "express-flash": "0.0.2",
15     "express-session": "^1.16.1",
16     "http-errors": "~1.6.2",
17     "jquery": "^3.4.0",
18     "md5": "^2.2.1",
19     "morgan": "^1.9.0",
20     "multer": "^1.4.1",
21     "mysql": "^2.17.1",
22     "parserjs": "^0.2.0",
23     "popper.js": "^1.14.7",
24     "pug": "^2.0.3",
25     "serve-favicon": "^2.5.0"
26   },
27   "devDependencies": {
28     "nodemon": "^1.18.11"
29   }
30 }
31

```

Nos pedirá un nombre, una versión del proyecto, descripción, si es privado o no, y un script el cual configuraremos para que arranque desde el fichero node.js al poner en la consola la palabra clave *"start"* que usaremos más adelante.

También nos da la opción de vincularlo directamente a un repositorio Git en la que se pide el tipo y una url. En mi caso como lo he estado manejando desde varios repositorios, lo he obviado.

Por último encontramos las dependencias. Aquí al comienzo posiblemente esté vacío, y nosotros a través de las librerías NPM que vayamos instalando se irán añadiendo automáticamente en nuestro package.json.

Nodemon hay que resaltar su utilidad, ya que va vinculado al proyecto a la hora de refrescar la aplicación cada vez que hay un cambio y un guardado en el código y nos lo notifica al momento, pudiendo así trabajar de forma más ágil.

Para instalarlo en el proyecto se ejecuta el siguiente comando:

```
npm install -g nodemon
```

Para que funcione nuestro servidor con Nodemon, vamos a realizar una modificación en el archivo package.json, cambiando *"start": "node index"* por el código *"start": "nodemon ./bin/www"*.

Finalmente, cuando queramos arrancar nuestro servidor solo tendremos que poner en la consola de Windows el siguiente comando e iniciará llamando a la configuración del package.json que hemos configurado:

```
npm start
```

```
C:\Users\acarras\INDECISION-CINEMA\INDECISION-CINEMA>npm start
> indecision-cinema@1.0.0 start C:\Users\acarras\INDECISION-CINEMA\INDECISION-CINEMA
> nodemon ./bin/www

[nodemon] 1.18.11
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node ./bin/www`
Connected as id 304
```

3.2.1 Express

Es una de las librerías más importantes de Node. Es aquella con la que podemos construir la aplicación de una forma más sencilla.

La forma de usarlo es instalarlo con el *npm install --save express* en consola y ya utilizarlo directamente en el código.

Una vez en el código, tendremos que importar la librería en nuestro file *app.js*

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

Después, nos crearemos una variable llamada *app*, con la cual iniciaremos Express y podremos usarla para inicializar todas variables de las librerías que hemos importado.

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

A partir de aquí, podremos usar los métodos de Express, como el método *use()* para poder utilizar todas las funcionalidades de las librerías que hayamos importado.

3.2.2 Instalación librerías NPM

Las diferentes librerías que podemos encontrar en la página oficial de NPM para Node ya en su propia definición y presentación de sus funcionalidades aparece cómo debemos instalarlas, de tal manera que se nos hace muy cómodo poder implementarlas fácilmente para su uso, aunque luego tendremos que configurarlas desde el código.

A la hora de la instalación, tendremos que poner el comando:

```
npm install libreriaNPMquequeramos
```

Cuando vayamos al código, en el *Package.json* aparecerá implementado directamente la librería/dependencia de Node que hemos instalado. De esta forma ahora solo hay que realizar las siguientes líneas de código para poder usarla.

Cogiendo el ejemplo de la librería *multer*, seguiremos los siguientes pasos:

- Tendremos que importar la librería en nuestro file *app.js*. Para ello usaremos la siguiente sentencia que irá al comienzo del file:

```
let multer = require('multer');
```

- Acto seguido, requerido el *multer* tendremos que usarlo. Para ello utilizaremos el método *use()*.


```
app.use(multer({
  storage: storage,
  dest: path.join(__dirname, 'public/images')
}).single('imagen'));
```

Y dentro añadiremos las funcionalidades que hagan falta. En este caso le estamos comunicando a multer que coja los datos que nos interesan y los guarde en un directorio de destino que nosotros le definimos.

En otros casos, no es necesario usar variables que requieran desde el código las librerías que importamos. Un ejemplo sería la librería de Bootstrap. En este caso, se instala igual desde la consola y el comando `npm install bootstrap`, pero a la hora de usarse, directamente configuramos con el método `use()` llamándolo desde la ruta que se encuentra dentro de nuestra carpeta directorio de dependencias de Node.

```
app.use('/bootstrap', express.static(__dirname + '/node_modules/bootstrap/dist')); // use bootstrap
```

Al ser una librería de JS para estilos, se añadirá posteriormente dentro del Header del .pug, para usarlo y dar el formato deseado.

3.3 MySQL y MySQL Workbench

La instalación de MySQL y componentes como Workbench es relativamente sencilla. Tendremos que entrar en la página oficial de MySQL y en el apartado de Downloads descargar la versión más moderna necesaria para nuestro SO de *MySQL Installer*. Una vez en tengamos en nuestro poder el ".exe" solo tendremos que ejecutarlo y seguir los pasos de la configuración.

Al llegar al apartado de configuración de la conexión a BBDD hay que tener en cuenta el puerto y crear un usuario con todos los privilegios necesarios.

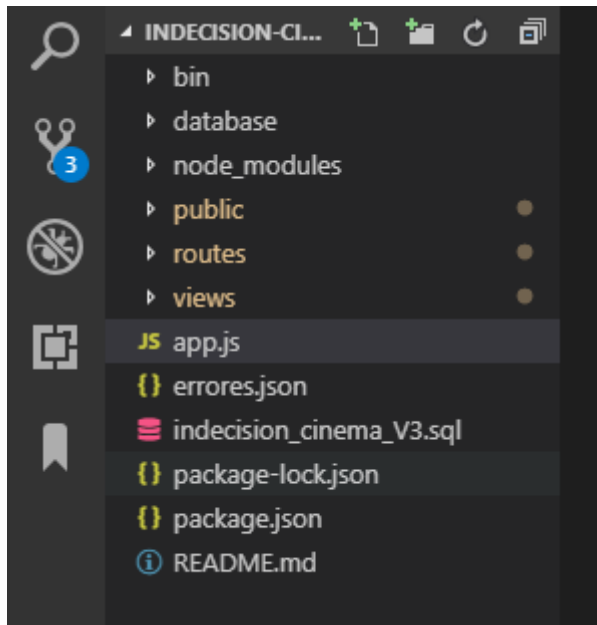
Desde MySQL Workbench podremos importar y exportar fácilmente la BBDD y realizar cambios, además de la gestión del diagrama entidad relación y diagramas de flujo.

4. Arquitectura de Proyecto

4.1 Estructura y Diseño

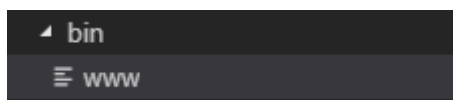
El proyecto tiene una estructura que se puede dividir en varias partes, las cuales se puede asemejar a un proyecto escrito en Java o en PHP, pero con algunas variantes.

A continuación las mencionamos:



4.1.1 bin

El directorio *bin* contiene a su vez el archivo *www* el cual se utiliza para arrancar la aplicación.



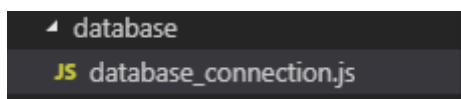
Si nos damos cuenta, a la hora de hacer un `npm start` en consola para arrancar el servidor y hacer correr la aplicación, nos aparece la ruta y un ID el cual varía con cada sesión.

```
C:\Users\acarras1\INDECISION-CINEMA\INDECISION-CINEMA>npm start
> indecision-cinema@1.0.0 start C:\Users\acarras1\INDECISION-CINEMA\INDECISION-CINEMA
> nodemon ./bin/www

[nodemon] 1.18.11
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node ./bin/www`
Connected as id 412
```

4.1.2 database

Es el directorio contenedor del archivo .js en el que tenemos la configuración necesaria para realizar la conexión a la Base de Datos.



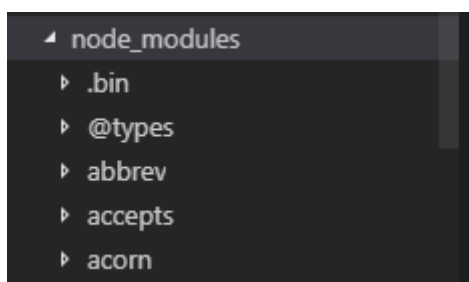
En el archivo database_connection.js importamos la librería de MySQL y creamos una variable de conexión adjuntando los datos necesarios. En nuestro caso son: host, database, user y password.

Adicionalmente se incluye una función que nos muestra por mensaje en consola si la conexión se ha realizado de forma correcta y nos mostraría el ID renovado con cada arranque del servidor, o si por el contrario ha habido algún fallo también nos lo avisaría.

```
1 let mysql = require('mysql');           // importar mysql
2 //const config = require();             // para fichero externo con las credenciales
3 //const dbConfig = config.db;
4
5 let connection = mysql.createConnection({
6   host: 'localhost',
7   database: 'indecision_cinema',
8   user: 'indecision_cinema',
9   password: 'totoro130891'
10 });
11
12 connection.connect(function(err) {
13   if (err) {
14     console.error('Error connecting: ' + err.stack);
15     return;
16   }
17   console.log('Connected as id ' + connection.threadId);
18 });
19
20 module.exports = connection;
```

4.1.3 node_modules

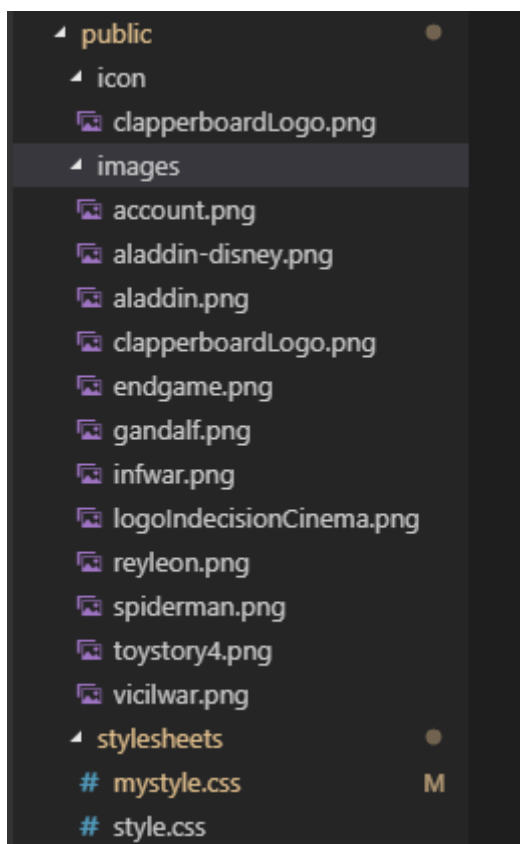
El directorio de *node_modules* es en el que se guardarán todas aquellas librerías de Node que nosotros vayamos instalando. Para poder usarlas, como se comentó en un punto anterior, se deberá importar y requerir esa librería para poder usarse más adelante.



En este directorio se encuentran tanto las librerías por defecto de Node, como las que vayamos añadiendo.

4.1.4 public

Public es el directorio en el cual se guardan otros directorios como *'icon'*, *'images'* y los *'stylesheets'* (estilos y CSS).



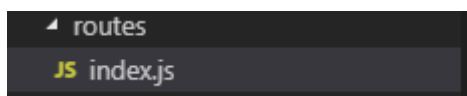
Para poder utilizarlos, hay que hacer utilizar el método `use()`, implementando en nuestro `app.js` la línea:

```
app.use(express.static(path.join(__dirname, 'public')));
```

De este modo, ya podremos utilizar el directorio sin problemas, guardando, sacando, utilizando información, etc.

4.1.5 routes

En este directorio guardaremos el archivo *index.js*. Es aquí donde se declararán y requerirán las rutas de las páginas o frames de pug además de las funciones necesarias para hacer la lógica del código.

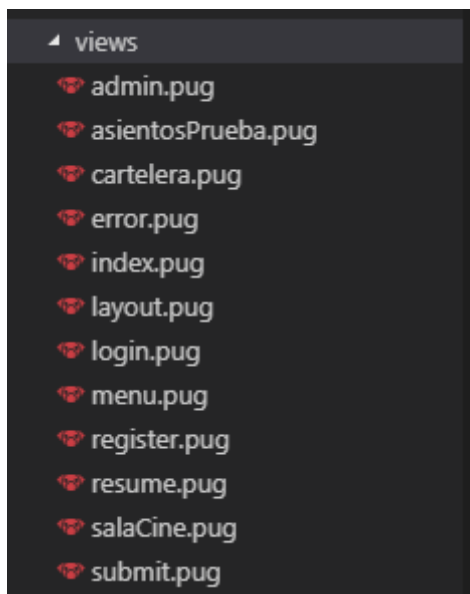


El código tiene su propio orden para poder funcionar. Primero se declaran las variables que guardan los requerimientos de las librerías, y después se realizan los `router.get()` con

la ruta del frame necesario o el `router.post()` que irán relacionadas con peticiones post seguidas de las funciones.






4.1.6 views

En el directorio Views se guardarán todas las páginas o frames .pug que serán el Front-end de la aplicación.



4.1.7 Resto de files

Fuera de directorios podemos encontrar en el proyecto los siguientes ficheros o files:

-  File `app.js`, donde se importan las librerías de las que se hacen uso y se declara e inicializa Express.
-  File `errores.json`, es un fichero en formato JSON en el que se implementó un error identificativo para llamarlo donde corresponde.
-  File `package-lock.json`, fichero en formato JSON donde se encuentra toda la información relacionada con el proyecto. Desde el nombre, versión, dependencias, etc.
-  File `package.json`, fichero en formato JSON donde se encuentran las configuraciones de forma resumida de los datos que nosotros implementamos a la hora de crear nuestro proyecto con Express.
-  Fichero `Readme.md`.

5. Desarrollo del Proyecto

A continuación se expondrán partes del código explicando su funcionalidad.

5.1 Conexión a BBDD

Para poder conectarnos a la base de datos, como explicamos más arriba hay que implementar el módulo de Node de MySQL, requerirlo y usarlo en el código.

Una vez hecha esta parte, dentro del folder *database* crearemos un fichero *database_connection.js* en el cual indicaremos los datos necesarios para poder conectarnos a BBDD.

```
database ▸ JS database_connection.js ▸ ...
1  let mysql = require('mysql');           // importar mysql
2
3  let connection = mysql.createConnection({
4    host: 'localhost',
5    database: 'indecision_cinema',
6    user: 'indecision_cinema',
7    password: 'totoro130891'
8  });
9
10 connection.connect(function(err) {
11   if (err) {
12     console.error('Error connecting: ' + err.stack);
13     return;
14   }
15   console.log('Connected as id ' + connection.threadId);
16 });
17
18 module.exports = connection;
```

Para poder exportar la variable y la conexión, indicaremos al final del fichero la línea *module.exports = connection;*

5.2 Login y creación de la Sesión

Para la creación de la Sesión utilizaremos la librería de Node llamada *express-session*, la cual instalaremos, crearemos la variable para requerirla y usarla:

```
let session = require('express-session');
```

Al usarla, podremos establecer tiempo de sesión por defecto para cuando el usuario esté inactivo durante un periodo de tiempo se desloguee automáticamente.

```
app.use(session({
  cookie: { maxAge: 100000 },
  store: sessionStore,
  saveUninitialized: true,
  resave: 'true',
  secret: 'secret'
}));
```

En cuanto a código en nuestro index.js implementaremos las routes con las funciones necesarias y la ruta hacia dónde se dirigirá la página una vez que el código llegue a ejecutarse.

Necesitaremos una función en la que recogemos los datos del formulario que hemos rellenado y comprobaremos con un SELECT si lo que nos devuelve tiene tamaño, en caso afirmativo creará la sesión guardada en la variable *loggedIn* y en el caso de que no se pueda crear la sesión se informará al usuario con un mensaje de error que dirá: *"Ha habido un problema con el usuario/contraseña"*

```
/**
 * Función para establecer la sesión
 */
router.post('/login', function (req, res) {
  let form = req.body;
  mysqlConnection.query("SELECT * FROM usuario WHERE email=? AND password=?", [form.email, md5(form.password)], function (err, result, fields) {
    if (err) throw err;
    if (result.length > 0) {
      req.session.loggedIn = true;
      req.session.admin = result[0].admin;
      req.session.email = result[0].email;
      req.session.nombre = result[0].nombre;
      req.session.apellidos = result[0].apellidos;
      res.redirect('/');
    } else {
      req.flash('error_messages', loginErr.userPass);
      res.redirect('/login');
    }
  });
  res.end();
});
```

Ya terminada la función, podremos redirigir a la página Home o Index donde aparecerá la opción de Logout en el navegador.

Las views del login se ven de la siguiente manera:

```
views > login.pug
1 extends menu
2
3 block content
4   body
5     main
6       #login
7       #logo
8       h1.tittle Log In
9       p.subtitle Te damos la bienvenida a INDECISION CINEMA
10      form(method='POST' action='/login')
11        .form-group
12          //label(for='email') Email
13          input.form-control(type='text' id='email' placeholder='email' name='email' required)
14        .form-group
15          //label(for='password') Password
16          input.form-control(type='password' id='password' placeholder='password' name='password' required)
17        button.btnLogin(type='submit') Log in
18        div#btnRegistrar
19          a.nav-regist(href="/register") Registrarte
20
21      if (error_messages.length > 0)
22        .alert.alert-danger #{error_messages}
```

Pug al ser un lenguaje basado en HTML pero indentado es bastante estricto para la nomenclatura. Aun así es generoso a la hora de dejarnos implementar lógica para complementar nuestra experiencia de usuario.

Podemos encontrar varios div con sus id y clases para poder darles formato en el CSS. El formulario y los input necesarios además del button que nos ayuda a registrar los datos del usuario y formalizar el proceso.

Abajo encontramos el control de errores.

5.3 Logout y destrucción de la Sesión

Este paso es más sencillo, usaremos el método *destroy()* para la sesión y redirigiremos la página a Home o Index. La función lo que hace es recoger la sesión la cual requerimos y aplicamos el método.

```
/**
 * Ruta de logout y destrucción de sesión
 */
router.get('/logout', function(req, res, next) {
  console.log(req.session);
  req.session.destroy();
  res.redirect('/');
});
```

5.4 Registro de Usuario

En este apartado tenemos dos métodos. El método *router.get()* que nos lleva a la página de registro con el formulario.

```
/* Ruta registro */
router.get('/register', function(req, res, next) {
  res.render('register', { title: 'Registrarse' });
});
```

Y otro método *router.post()* el cual contiene una función que hace la comparación en la BBDD para comprobar si el email (PK) está registrado en la tabla Usuarios. En el caso de que el email esté ya registrado, sacará un mensaje de error informando al usuario. En caso de que sea un registro nuevo hará el insert e informará al usuario con un mensaje de que todo ha ido Ok.


```

/**
 * Función para comprobación de email duplicado e inserción de registro en BBDD
 */
router.post('/register', function (req, res) {
  let email = req.body.email;
  let sqlCheckEmail = "SELECT * from usuario WHERE email = ?";

  mysqlConnection.query(sqlCheckEmail, [email], function (err, rows) {
    if (rows.length) {
      console.log(this.sql);
      req.flash('error_messages', 'Email ya registrado. Prueba con otro :');
      res.redirect('/register');
    } else {
      let nombre = req.body.name;
      let apellidos = req.body.surname;
      let dni = req.body.dni;
      let password = md5(req.body.password);
      let admin = 1;
      let sql = "INSERT INTO usuario (email, dni, nombre, apellidos, admin, password) VALUES (?, ?, ?, ?, ?, ?)";

      mysqlConnection.query(sql, [email, dni, nombre, apellidos, admin, password], function (err, result) {
        if (err) {
          console.log(this.sql);
          throw err;
        } else {
          req.flash('success_messages', 'Se ha registrado correctamente. Encontrarás en tu correo todos los datos.');
```

5.5 Cartelera

En la cartelera necesitamos tener la ruta desde el *router.get()* en la que añadiremos una función que nos ejecute un SELECT. Esa query nos sacará de base de datos todos los elementos que necesitaremos para mostrarlos después.

```

/**
 * Ruta de Cartelera
 */
router.get('/cartelera', function(req, res, next) {
  mysqlConnection.query("select titulo, duracion, descripcion, imagen, num_sala, fecha_reproduccion, hora_reproduccion from pelicula, "+
    "sala where sala.titulo=pelecula.titulo AND enCartelera=0", function (err, results, fields) {
    if (err) throw err;
    res.render('cartelera', { title: 'Cartelera', peliculas: results});
    res.end();
  });
});

```

En pantalla las películas en cartelera. Todos los datos recogidos se los pasaremos por parámetros al *.pug*

```

views > cartelera.pug
1 | extends menu
2 |
3 | block content
4 |   body
5 |     main
6 |       #cartelera
7 |         #titulo
8 |         h2 Cartelera
9 |         #panelPeliculas
10 |         each pelicula in peliculas
11 |           a.nav-caratula(href="salaCine?pelicula="+pelicula.titulo)
12 |             .caratula(style="background-image: url(public/images/'+pelicula.imagen)
13 |               .descripcion
14 |               h3 #{pelicula.titulo}
15 |               p.desCaratula #{pelicula.descripcion}
16 |               p.desCaratula Número de Sala: #{pelicula.num_sala}
17 |               p.desCaratula Duración en min: #{pelicula.duracion}
18 |               //p.desCaratula Fecha Reproducción: #{pelicula.fecha_reproduccion.toDateString()}
19 |               p.desCaratula Fecha Reproducción: #{pelicula.fecha_reproduccion.toDateString()}
20 |               p.desCaratula Hora Reproducción: #{pelicula.hora_reproduccion}

```

En la imagen se muestra cómo pasar los parámetros sacados del array de datos que tomamos con los resultados de la query SELECT.

5.6 Funciones Admin

Como administrador del cine, el usuario encargado tendrá la función de crear y borrar las salas y películas que mostrará al público para su venta.

5.6.1 Creación y Borrado de Salas

Para registrar las salas, se usan las *router.post()* con la función correspondiente. Para ello declaramos la función y le pasamos el req, que es el requerimiento, y el res, que es el resultado que nos dará la función. Una vez dentro de la función, nos declaramos las variables en las cuales inicializamos con los datos que recogemos del formulario.

Una vez realizado este paso, nos creamos la query la cual pasaremos junto con los datos a recoger y comprobamos que se ha guardado correctamente en la base de datos.

```
/**
 * Función RegistrarSala, encargada de hacer un INSERT en la BBDD en la tabla SALA
 */

router.post('/registrarSala', function registrarSala (req, res) {
  let num_sala = req.body.num_sala;
  let fecha_repro = req.body.fecha_repro;
  let hora_repro = req.body.hora_repro;
  let titulo_peli = req.body.titulo_peli;
  let sql = "INSERT INTO sala (num_sala, fecha_reproduccion, hora_reproduccion, titulo_peli) VALUES (?, ?, ?, ?)";

  mysqlConnection.query(sql, [num_sala, fecha_repro, hora_repro, titulo_peli], function (err, result) {
    if(err){
      console.log(this.sql);
      throw err;
    } else {
      req.flash('success_messages', 'se ha guardado en bdd');
      res.redirect('/admin');
    }
  });
  res.end();
});
});
```

En el caso del formulario del .pug, cogeremos los datos que hemos metido en el input.

```
#nuevaSala
p Publicar una sala para la película en cartelera
form(method='POST' action='/registrarSala')
  .form-group
    input.form-control(type='num' id='num_sala' placeholder='numero sala' name='num_sala' required)
  .form-group
    input.form-control(type='num' id='titulo_peli' placeholder='titulo_peli' name='titulo_peli' required)
  .form-group
    input.form-control(type='date' id='fecha_repro' placeholder='fecha_repro' name='fecha_repro' required)
  .form-group
    input.form-control(type='num' id='hora_repro' placeholder='Ej 10:00:00' name='hora_repro' required)
  button.btnLogin(type='submit') Crear
```

Todos los campos son requeridos.

Para el efectuar el borrado de la sala, tendremos que recoger los datos desde el formulario. Lo que realizamos es crearnos una variable con el id de sala (*num_sala*) y la query correspondiente para luego ejecutarla comparando con el registro del id de sala. Una vez realizado de forma eficiente, se lanzará un mensaje satisfactorio. Además se ha implementado una segunda query que se ejecuta justamente después, borrando los registros comprados u ocupados de la sala para que la próxima vez que el administrador quiera reabrir una sala en concreto, la sala quede nuevamente vacía para su uso y venta.

```

/**
 * Función Borrar Sala de Cine. Encargada de hacer un DELETE de la tabla de BBDD
 */
router.post('/borrarSala', function borrarSala (req, res) {
  console.log('INIT BorrarSala -----');
  let num_sala = req.body.num_sala;
  let sql = "DELETE FROM sala WHERE num_sala=?";
  let sqlborrarasientos = "Update asientos Set ocupado='0' Where ocupado='1' and num_sala=? ";

  mysqlConnection.query(sql, [num_sala], function (err, result) {
    if(err){
      console.log(this.sql);
      req.flash('error_messages', 'ERROR - No se ha podido borrar.');
```

El formulario de borrado no dista del resto, tiene declarado el input con el botón que hace el ejecutado de la acción.

```

#borrarPeli
p Borrar Pelicula en Cartelera
form(method='POST' action='/borrarPelicula')
  .form-group
    input.form-control(type='text' id='titulo' placeholder='titulo' name='titulo' required)
    button.btnBorrar(type='submit') Borrar

```

5.6.2 Creación y Borrado de Películas

Para registrar una película en BBDD para poder después asignarla a una Sala y publicarla en cartelera, primero debemos subir la imagen en un formulario aparte. Es necesario para comodidad y control del administrador tener las imágenes que serán de la cartelera guardadas en su carpeta /images del proyecto.

De modo que primero se efectúa la subida de la imagen para poder después asignar esa imagen a la película que a continuación asignaremos a la sala que queramos.

```

// route MULTER
app.use(multer({
  storage: storage,
  dest: path.join(__dirname, 'public/images')
}).single('imagen'));

app.post('/subirImagen', (req, res) => {
  //console.log(req.file);
  req.flash('success_messages', 'Se ha añadido correctamente.');
```

Y el formulario se vería de la siguiente manera:

```
#nuevaImgCartelera
p Sube una nueva imagen de cartelera a tu cine
  form(method='POST' action='/subirImagen' enctype='multipart/form-data')
    .form-group
      input.form-control(type='file' id='imagen' placeholder='imagen' name='imagen' required)
    button.btnLogin(type='submit') Subir
p2 Recuerda que la imagen debe tener un tamaño de 300x400px
```

Como características a tener en cuenta, *multer* que es la librería usada para poder realizar la subida de ficheros (imágenes en este caso), trabaja por libre, es decir, tiene su propio *enctype*.

Para registrar al película ejecutaremos un *routes.post()* con su función, variables declaradas y query requerida con su INSERT, y se realizará de la misma forma con el DELETE filtrando por el título, el cual es el id de la película que queramos.

```
/**
 * Función Registrar Película se encarga de hacer un INSERT de la tabla de BBDD
 */
router.post('/registrarPelícula', function registrarPelícula (req, res) {
  console.log('INIT RegistrarPelícula -----');
  let titulo = req.body.titulo;
  let duracion = req.body.duracion;
  let descripcion = req.body.descripcion;
  let imagen = req.body.imagen;
  let sql = "INSERT INTO pelicula (titulo, duracion, descripcion, imagen) VALUES (?, ?, ?, ?)";

  mysqlConnection.query(sql, [titulo, duracion, descripcion, imagen], function (err, result) {
    if(err) {
      console.log(this.sql);
      throw err;
    } else {
      req.flash('success_messages', 'Se ha añadido correctamente.');
```

En el formulario haremos una recogida de datos para ambos casos, quedando de la siguiente manera:

```
#nuevaSala
p Publicar una sala para la película en cartelera
  form(method='POST' action='/registrarSala')
    .form-group
      input.form-control(type='num' id='num_sala' placeholder='numero sala' name='num_sala' required)
    .form-group
      input.form-control(type='num' id='titulo_peli' placeholder='titulo_peli' name='titulo_peli' required)
    .form-group
      input.form-control(type='date' id='fecha_repro' placeholder='fecha_repro' name='fecha_repro' required)
    .form-group
      input.form-control(type='num' id='hora_repro' placeholder='Ej 10:00:00' name='hora_repro' required)
    button.btnLogin(type='submit') Crear

#borrarSala
p Borrar Sala
  form(method='POST' action='/borrarSala')
    .form-group
      input.form-control(type='num' id='num_sala' placeholder='numero sala' name='num_sala' required)
    button.btnBorrar(type='submit') Borrar
```

5.6.3 Selección y actualización de asientos

Sin lugar a dudas, la parte más compleja del proyecto se encuentra en esta parte. Para que llegase a funcionar, se recogen desde el .pug los datos con el método `.click()` al pinchar sobre el asiento o asientos requeridos. Y se ejecutan métodos que nos cambian los estados para que el usuario desde la vista pueda apreciar que el asiento seleccionado ha cambiado su estado de *libre* ha seleccionado.

```

1  extends menu
2
3  block content
4    if (session.loggedin)
5      h2 Sala Cine
6      .btn.btn-primary.comprar Comprar
7      main#salaCine(data-numSala=numSala)
8        - var fila = 5
9        - var butaca = 1
10       each val in [5, 4, 3, 2, 1]
11         p.numFila: span= val
12         #asientos
13         each asiento in asientos
14           if (fila == asiento.fila)
15             if (butaca == asiento.butaca)
16               if (asiento.ocupado === 0)
17                 .libre(data-id=asiento.id_asiento): span= butaca
18               else
19                 .ocupado: span= butaca
20                 - butaca++
21             else
22               - butaca= 1
23               if (asiento.ocupado === 0)
24                 .libre(data-id=asiento.id_asiento): span= butaca
25               else
26                 .ocupado: span= butaca
27                 - butaca++
28         - fila--
29       #pantalla: span pantalla
30       #leyenda
31       p Libre
32       .libre
33       p Ocupado
34       .ocupado
35       p Seleccionado
36       .seleccionado
37     else
38       #noLogueado
39       p Warning - Tienes que loguearte antes de poder acceder a la sala.
40       #imagenWarning
41

```

A la hora de pintar los asientos, intercambiamos los datos y creamos bucles para que nos pinten las filas de los asientos como nosotros pidamos, dándoles así un ID a cada asiento para poder darle formato y trabajar con él.

Lo que estamos realizando con el siguiente código, es crearnos un array de asientos. Cambiaremos el estado del `div` de asientos seleccionado con el método `click()`, no usaremos el método `onClickListener()` debido a que puede crear multipeticiones y sobrecargar el flujo.

```

script.
let id_asientos = [];

$(".libre").click(function(event) {
  setSelectedSeats($(this), event);
});

$(".comprar").click(function(event) {
  comprarAsientos();
});

function setSelectedSeats(element, event){
  const dataId = "data-id";
  const classSelected = "seleccionado";

  if(element.hasClass(classSelected)){
    removeSeat(getDataIdAttribute(event, dataId))
  } else {
    id_asientos.push(getDataIdAttribute(event, dataId));
  }
}

function getDataIdAttribute(event, attr){
  return $(event.currentTarget).attr(attr);
}

function comprarAsientos(){
  if (id_asientos.length > 0) {
    window.location.href = window.location.origin + "/purchaseFilm?numSala=" + $("#salaCine").attr("data-numsala") + "&id_asientos=" + id_asientos;
  }
}

function removeSeat(id){
  var index = id_asientos.indexOf(id);
  if (index > -1) {
    id_asientos.splice(index, 1);
  }
}

```

Resumiendo, lo que el código realiza en las funciones que se visualizan un poco más arriba es recoger los eventos y hacer los cambios de estado relacionándolos con el ID del asiento seleccionado.

Una vez llamen a la función *comprarAsientos()*, contiene una condición que mientras nos devuelva valor recogido de los asientos, nos enviará junto con el SELECT que tenemos en la función de index.js hasta la página de Resume con los datos de la compra.

```

/**
 * Función GET RESUME encargada de recoger los datos finales del Resumen
 */
router.get('/resume', function(req, res) {
  var url_parts = url.parse(req.url, true);
  var query = url_parts.query;

  let asientos = query.asientos;
  let numSala = query["num_sala"];

  let sqlSelect = "SELECT pelicula.titulo as titulo, pelicula.imagen as imagen, pelicula.duracion as duracion, sala.num_sala as sala, sala.fecha_reproduccion as fecha, "+
    "sala.hora_reproduccion as hora, asientos.fila as fila, asientos.butaca as butaca FROM pelicula, sala, asientos " +
    "WHERE pelicula.titulo = sala.titulo_peli AND sala.num_sala = asientos.num_sala AND sala.num_sala=" + numSala + " AND asientos.id_asiento IN(" + asientos + ")";

  mysqlConnection.query(sqlSelect, [], function (err, datos) {
    if (err) throw err;

    res.render('resume', { datos : datos[0], seats : datos});
    res.end();
  });
});

```

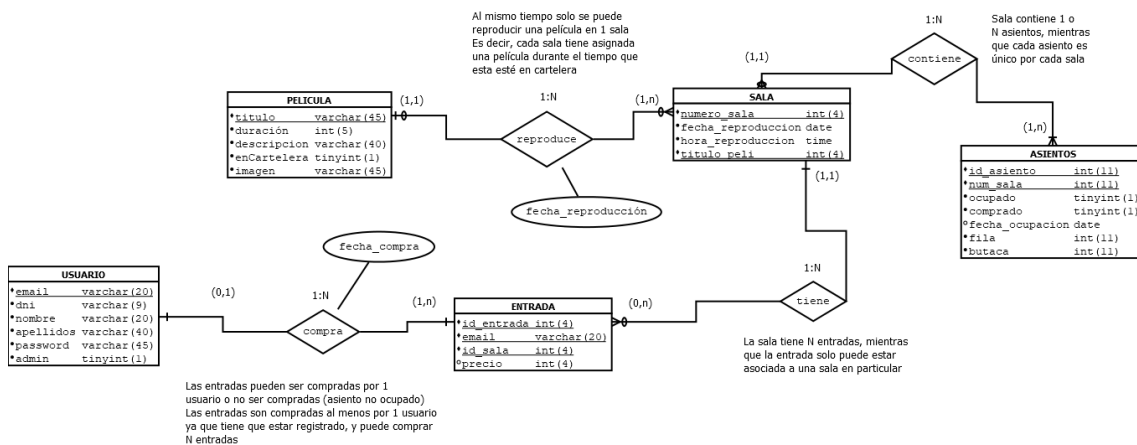
6. Desarrollo de la BBDD

En este punto se documentará todo lo referido con la Base de Datos de Indecision Cinema.

La base de datos se comprenderá de 5 tablas: *asientos*, *entrada*, *película*, *sala* y *usuario*.

```
mysql> show tables;
+-----+
| Tables_in_indecision_cinema |
+-----+
| asientos                     |
| entrada                     |
| pelicula                    |
| sala                        |
| usuario                     |
+-----+
5 rows in set (0.07 sec)
```

6.1 Diagrama Entidad Relación



- Las entradas pueden ser compradas por 1 usuario o no ser compradas (asiento no ocupado). Las entradas son compradas al menos por 1 usuario ya que tiene que estar registrado, y puede comprar N entradas.

USUARIO compra ENTRADA -> 1:N

- La sala tiene N entradas, mientras que la entrada solo puede estar asociada a una sala en particular.

ENTRADA tiene SALA -> 1:N

- Sala contiene 1 o N asientos, mientras que cada asiento es único para cada sala.

SALA contiene ASIENTOS -> 1:N



Al mismo tiempo solo se puede reproducir una película en 1 sala, es decir, cada sala tiene asignada una película durante el tiempo que esta esté en cartelera.
SALA reproduce PELICULA -> 1:N

6.2 Script

Comienza con la creación de la BBDD si no existe, e indicación de que debe usarse.

```
CREATE DATABASE IF NOT EXISTS `indecision_cinema`  
USE `indecision_cinema`;
```

Creación de tabla Asientos.

```
DROP TABLE IF EXISTS `asientos`;  
/*!40101 SET @saved_cs_client = @@character_set_client */;  
SET character_set_client = utf8mb4 ;  
CREATE TABLE `asientos` (  
  `id_asiento` int(11) NOT NULL AUTO_INCREMENT,  
  `ocupado` tinyint(1) NOT NULL DEFAULT '0',  
  `num_sala` int(11) NOT NULL,  
  `comprado` tinyint(1) NOT NULL DEFAULT '0',  
  `fecha_ocupacion` date DEFAULT NULL,  
  `fila` int(11) NOT NULL,  
  `butaca` int(11) NOT NULL,  
  PRIMARY KEY (`id_asiento`,`num_sala`),  
  KEY `fk_ASIENTOS_SALA1_idx` (`num_sala`)  
) ENGINE=InnoDB AUTO_INCREMENT=61 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Ejemplo inserción de datos en la tabla Asientos (largo INSERT...)

```
LOCK TABLES `asientos` WRITE;  
/*!40000 ALTER TABLE `asientos` DISABLE KEYS */;  
INSERT INTO `asientos` VALUES (1,1,1,0,NULL,1,1),(1,0,2,0,NULL,1,1),(1,0,3,0,NULL,1,1),(1,0,4,0,NULL,1,1),  
/*!40000 ALTER TABLE `asientos` ENABLE KEYS */;  
UNLOCK TABLES;
```

Creación de la tabla Entrada:

```
DROP TABLE IF EXISTS `entrada`;  
/*!40101 SET @saved_cs_client = @@character_set_client */;  
SET character_set_client = utf8mb4 ;  
CREATE TABLE `entrada` (  
  `id_entrada` int(11) NOT NULL AUTO_INCREMENT,  
  `email` varchar(45) NOT NULL,  
  `precio` int(11) DEFAULT NULL,  
  `fecha_compra` date NOT NULL,  
  PRIMARY KEY (`id_entrada`,`email`),  
  UNIQUE KEY `id_entrada_UNIQUE` (`id_entrada`),  
  UNIQUE KEY `email_UNIQUE` (`email`),  
  KEY `fk_ENTRADA_USUARIO1_idx` (`email`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```


Creación de la tabla Película:

```
DROP TABLE IF EXISTS `pelicula`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
SET character_set_client = utf8mb4 ;
CREATE TABLE `pelicula` (
  `titulo` varchar(45) NOT NULL,
  `duracion` int(11) NOT NULL,
  `descripcion` varchar(100) NOT NULL,
  `enCartelera` int(11) NOT NULL DEFAULT '0',
  `imagen` varchar(45) NOT NULL,
  PRIMARY KEY (`titulo`),
  UNIQUE KEY `titulo_UNIQUE` (`titulo`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Ejemplo inserción en la tabla película:

```
LOCK TABLES `pelicula` WRITE;
/*!40000 ALTER TABLE `pelicula` DISABLE KEYS */;
INSERT INTO `pelicula` VALUES ('Aladdin',180,'Aladdin y la lámpara mágica',0,'aladdin-disney.png'),
/*!40000 ALTER TABLE `pelicula` ENABLE KEYS */;
UNLOCK TABLES;
```

Creación tabla Sala:

```
DROP TABLE IF EXISTS `sala`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
SET character_set_client = utf8mb4 ;
CREATE TABLE `sala` (
  `num_sala` int(11) NOT NULL,
  `fecha_reproduccion` date NOT NULL,
  `hora_reproduccion` time NOT NULL,
  `titulo_peli` varchar(45) NOT NULL,
  PRIMARY KEY (`num_sala`),
  UNIQUE KEY `num_sala_UNIQUE` (`num_sala`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Ejemplo inserción tabla Sala:

```
LOCK TABLES `sala` WRITE;
/*!40000 ALTER TABLE `sala` DISABLE KEYS */;
INSERT INTO `sala` VALUES (1,'2019-06-04','11:00:00','Aladdin'),
/*!40000 ALTER TABLE `sala` ENABLE KEYS */;
UNLOCK TABLES;
```

Creación tabla Usuario:

```
DROP TABLE IF EXISTS `usuario`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
SET character_set_client = utf8mb4 ;
CREATE TABLE `usuario` (
  `email` varchar(45) NOT NULL,
  `dni` varchar(45) NOT NULL,
  `nombre` varchar(45) NOT NULL,
  `apellidos` varchar(45) NOT NULL,
  `admin` tinyint(1) NOT NULL DEFAULT '0',
  `password` varchar(45) NOT NULL,
  PRIMARY KEY (`email`),
  UNIQUE KEY `email_UNIQUE` (`email`),
  UNIQUE KEY `dni_UNIQUE` (`dni`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Ejemplo inserción en tabla Usuario - Admin:

```
LOCK TABLES `usuario` WRITE;
/*!40000 ALTER TABLE `usuario` DISABLE KEYS */;
INSERT INTO `usuario` VALUES ('admin@gmail.com','87776372Ñ','admin','admin',1,'admin'),(
/*!40000 ALTER TABLE `usuario` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;
```







Ejemplo inserción en tabla Usuario – Usuario normal:

```
,('aliss1391@gmail.com','52008850P','Alicia','Carrascoso Lozano',0,'c9cb0d28f755b1b2e61ca14ffef0334d')
```

6.3 Tablas

Descripción y detallado de las tablas de la Base de Datos.

6.3.1 Usuario

-  email varchar (45) PK U: que será el ID del usuario.
-  dni varchar (45) U: será único.
-  nombre varchar(45)
-  apellidos varchar(45)
-  admin tinyint (1): será booleano. 1 es admin, 0 es usuario al uso.
-  Password varchar (45): será cifrada a la hora de registrarla en BBDD con md5.

```
mysql> describe usuario;
```

Field	Type	Null	Key	Default	Extra
email	varchar(45)	NO	PRI	NULL	
dni	varchar(45)	NO	UNI	NULL	
nombre	varchar(45)	NO		NULL	
apellidos	varchar(45)	NO		NULL	
admin	tinyint(1)	NO		0	
password	varchar(45)	NO		NULL	

```
6 rows in set (0.02 sec)
```

6.3.2 Película

- 📺 titulo varchar (45) PK U: será el ID de película
- 📺 duración int (11)
- 📺 descripción varchar (45)
- 📺 enCartelera int (11): será booleano, 1 no está en cartelera, 0 si está en cartelera.
- 📺 Imagen varchar (45): se guarda la ruta de la imagen desde el folder /public/image

```
mysql> describe pelicula;
```

Field	Type	Null	Key	Default	Extra
titulo	varchar(45)	NO	PRI	NULL	
duracion	int(11)	NO		NULL	
descripcion	varchar(100)	NO		NULL	
enCartelera	int(11)	NO		0	
imagen	varchar(45)	NO		NULL	

```
5 rows in set (0.01 sec)
```

6.3.3 Sala








- 📺 num_sala int (11) PK: será la Primary Key pero no será unique, ya que se usará el número de sala para introducir todos los asientos posteriormente en la tabla Asientos.
- 📺 fecha_reproduccion date
- 📺 hora_reproduccion time
- 📺 titulo_peli varchar (45) será lo que utilizemos para enlazar más adelante las tablas de sala y película.

```
mysql> describe sala;
```

Field	Type	Null	Key	Default	Extra
num_sala	int(11)	NO	PRI	NULL	
fecha_reproduccion	date	NO		NULL	
hora_reproduccion	time	NO		NULL	
titulo_peli	varchar(45)	NO		NULL	

```
4 rows in set (0.00 sec)
```

6.3.4 Asientos





-  id_asiento int (11) PK Autoincrementada.
-  ocupado tinyint (1) booleano
-  num_sala int (11) FK para relacionar los asientos con la sala
-  comprado tinyint (1) booleano
-  fecha_ocupacion date
-  fila int (11)
-  butaca (11)

```
mysql> describe asientos;
```

Field	Type	Null	Key	Default	Extra
id_asiento	int(11)	NO	PRI	NULL	auto_increment
ocupado	tinyint(1)	NO		0	
num_sala	int(11)	NO	PRI	NULL	
comprado	tinyint(1)	NO		0	
fecha_ocupacion	date	YES		NULL	
fila	int(11)	NO		NULL	
butaca	int(11)	NO		NULL	

```
7 rows in set (0.00 sec)
```

6.3.5 Entrada

-  id_entrada int (11) PK U Autoincrementada.
-  email varchar (45) FK de Usuarios.
-  precio int (11)
-  fecha_compra date

```
mysql> describe entrada;
```

Field	Type	Null	Key	Default	Extra
id_entrada	int(11)	NO	PRI	NULL	auto_increment
email	varchar(45)	NO	PRI	NULL	
precio	int(11)	YES		NULL	
fecha_compra	date	NO		NULL	

```
4 rows in set (0.00 sec)
```

7. Pruebas

Se crearán distintos escenarios para las pruebas, testeando la aplicación a prueba de fallos comunes o más enrevesados y comprobando los resultados obtenidos:

Escenario	Resultado Esperado	Resultado Obtenido	Solución
Primera vez que se prueba la conexión a BBDD	Devolver un id de sesión por consola para la comprobación de la escucha del servidor en el puerto 3000	Módulo Mysql no encontrado	Incorporar la librería de node de mysql aparte de tener mysql instalado. Y creación de la variable requerida.
Loguear usuario con email y contraseña desde la página de Login	Guardar la session en la variable loggedin y efectuar el cambio a ruta de home	Contraseña mal metida o email incorrecto	Añadir un control de errores para la comprobación del usuario
Registro de usuario con email duplicado desde la página de Registro	Error por consola	Página 404 not found	Incorporar un control para si el email esta duplicado, salga mensaje al usuario
Página de administración para insertar los campos en BBDD de películas	Insertión hecha de forma satisfactoria	Campos encontrados a null	Interferencias con el enctype requerido para la subida del file imagen al proyecto, por lo que tuve que hacerlo en dos partes para que no hubiese confrontaciones de código en la llamada.
Mostrar la vista Sala de Cine con los asientos	Visualización correcta	Se muestran aunque el usuario esté deslogueado, cuando para comprar es necesario estar logueado.	Implementar en el pug un control comprobando si la variable loggedin contiene datos de session.
Mostrar la vista Cartelera	Que se muestre correctamente	Error 404 not found	Corrección del identado del código, añadiendo las variables requeridas con la nomenclatura necesaria.
Mostrar la vista Resume de la compra del usuario al darle al botón "comprar"	Que se muestre correctamente	Numero de sala undefined	Necesaria investigación de jQuery y JavaScript para poder utilizar el debug en caliente sobre el navegador e implementación de las variables con la url adecuada en el window.location.href
Hacer clic para seleccionar asiento y	Funcionalidad correcta	Selecciona el asiento correctamente, pero no lo deselectiona	Implementación del método click con el evento que borre el

volver a seleccionar para dejarlo libre			estado del asiento recogiendo el ID en un array y borrando la posición
Al hacer el select de la sala con los datos	Visualización correcta	El campo fecha se veía con otro formato	Cambio e implementación en el Pug del método .toDateString() en la fecha
Mostrado de datos en vista del resumen de compra	Visualización correcta	Error por consola en las querys	Separando las querys de UPGRADE y SELECT en dos funciones distintas y llamadas de forma ordenada.

8. Conclusión

En este apartado me dirigiré en primera persona para explicar de primera mano todas las conclusiones, paradojas, dificultades, retos y logros que he podido obtener gracias al proyecto final de ciclo.

8.1 Dificultades

En este proyecto me he encontrado con bastantes piedras en el camino. Entre ellas podría destacar el estudio e implementación del proyecto en un lenguaje moderno y novedoso en continuo cambio como es Nodejs. Además, sus librerías NPM aunque sean fáciles de instalar, presentaban sus diferentes metodologías de uso que a la hora de implementar no casaban perfectamente con la arquitectura del proyecto, teniendo que hacer modificaciones en caliente una vez las funcionalidades estaban prácticamente acabadas.

Por otro lado, MySQL y sus versiones no siempre funcionan a la primera, y a la hora de pasarme el proyecto por un repositorio Git a través de GitKraken para poder trabajar desde cualquier máquina, tenía el problema de que la versión de mysql no era siempre la adecuada y había que reajustar de nuevo la configuración.

No solo he tenido que familiarizarme con Node, también he usado otros lenguajes que no estaban en mi conocimiento, como por ejemplo Pug, que es un lenguaje de marcas basado en HTML pero indentado y presentado sin etiquetado de un formato muy estricto.

Además, finalmente tuve que implementar una variante de jQuery para poder llevar a cabo la lógica de los asientos y sus estados, además de la redirección a la página de resumen de compra.

Como resumen, las mayores dificultades fueron al comienzo del proyecto para entender su estructura y cómo implementar todas sus funcionalidades, y luego a la hora de desarrollar, el enfrentarme a una investigación y comprensión de un nuevo lenguaje de programación para llevar a cabo el proyecto.

8.2 Alcance y Limitaciones

El alcance de este proyecto, al ser un cine y desarrollado en un lenguaje tan potente, es de una gran envergadura. El alcance inicial fue posiblemente mayor del que podía abarcar y por tanto tuve muchas limitaciones a la hora de realizarlo.

A grandes rasgos el alcance del proyecto ha llegado a poderse realizar con las funcionalidades básicas de un cine hasta el momento de la compra.

Las limitaciones principales que se han encontrado han sido sobre todo el tiempo disponible dedicado y las librerías adjuntas a Node con su propia configuración base. Además, incluir las funcionalidades sobre el código no es tan semejante a otros lenguajes de programación y eso ha sido causa de muchos errores y pruebas a lo largo del desarrollo del proyecto.

Otra de las limitaciones más importantes a la hora de realizar el desarrollo del proyecto ha sido el tema de hacer *debug* al código. Nodejs no tiene un modo claro de debuguear aparte del uso de los `console.log ()` y el inspeccionar del navegador.
















Además, uno de los objetivos fue poder usar el modelo MVP (Model View Presenter), pero al final para poder sacar el código a flote no se ha podido cumplir al 100%.

8.3 Líneas futuras de investigación


El proyecto en un principio se intentó que fuese incluyendo todas las funcionalidades posibles abarcables en el tiempo disponible para realizarse, pero finalmente debido a las investigaciones y tiempo dedicado para comprender, aprender y afianzar conocimientos se tuvo que recortar el alcance del proyecto.

Por ello, en vistas a futuro las líneas de investigación son muy extensas, pudiendo desarrollar el proyecto de una forma inmensa.

Podríamos destacar:

-  Implementación de un email que se envíe al registrar al usuario con todos los datos de su cuenta.
-  Implementación de un "*recordar contraseña*" si por algún motivo el usuario intentase loguearse pero no recuerda sus credenciales.
-  Implementación de un aviso al usuario cuando se desloguee por tiempo de sesión expirado o porque él mismo lo desee.
-  Implementación de publicidad para que la página tuviese ingresos.
-  Implementación de una vista más atractiva y dinámica del Home con carroussel y noticias de otras páginas relacionadas con el mundo del cine.
-  Hacer que "*Los próximos estrenos*" de la página Home fuesen dinámicos y se recogiesen por las cookies de otras páginas o por otro lado podría ser el propio administrador el que gestionase la página web al completo.
-  Implementación de Salas con varias sesiones, ya que actualmente solo se puede tener 1 sesión de 1 película concreta para 1 día concreto.
-  Implementación de un formato más visual en las carteleras.
-  Creación de asientos especiales para gente con movilidad reducida.
-  Implementación del precio dinámico de las entradas con descuentos según edades o carné de socio/joven/etc.
-  Implementación de un carro de compra efectivo y funcional.
-  Implementación de un código QR para descargar las entradas al momento de efectuarse la factura.
-  Implementar dar opciones al usuario para efectuar los pagos, vía PayPal o Tarjeta de Débito/Crédito.
-  Implementar la opción de descargar las entradas en formato PDF al email del usuario registrado que ha efectuado la compra.
-  Implementación de la opción con preventa.

En cuanto a otras funcionalidades, también podrían añadirse:

-  Guardar los asientos primero como reservados con la fecha y hora del momento y con la compra ya efectiva cambiar el estado a asiento ocupado.

- 🏗️ Implementación correcta del modelo MVP (Model View Presenter).
- 🏗️ **Por último y no menos importante:** La funcionalidad por la que Node fue elegido para este proyecto pero que no se pudo realizar, y es usar la concurrencia para con multi-hilos poder crear reservas sincronizadas de los asientos.

















8.4 Conclusión

Indecision Cinema es un Proyecto Final para el Grado Superior de DAIM (*Desarrollo Aplicaciones Informáticas Multiplataforma*) construido con una tecnología puntera como es Nodejs con Express para aplicación Web corriendo en servidor, con un back-end formado por una composición de librerías de Node muy extensa y un front-end basado en las tecnologías más potentes en la actualidad a nivel de respuesta. Por este motivo es un proyecto muy ambicioso del que creo que he podido sacar muchas de las funcionalidades esperadas de una forma más que aceptable para el tiempo disponible y dedicado para llevarlo a cabo.

Sobre todo, me quedo con la experiencia de enfrentarme a un lenguaje nuevo y poder llegar a comprender su funcionamiento e implementarlo en una idea que viene sobre todo de las ganas de crear una aplicación visualmente agradable y dinámica, con muchas vistas y desarrollo a futuro disponible.

9. Bibliografía

A continuación pondré a disposición los canales que he consultado para poder llevar a cabo este proyecto:

-  Node tutoriales: <https://www.tutorialspoint.com/nodejs/index.htm>
-  Node W3Schools: <https://www.w3schools.com/nodejs/default.asp>
-  Node Descarga y Documentación: <https://nodejs.org/es/>
-  Librerías NPM: <https://www.npmjs.com/>
-  Express: <https://expressjs.com/es/>
-  Express tutoriales: <https://expressjs.com/es/starter/generator.html>
-  MySQL: <https://www.mysql.com/>
-  Consulta queries MySQL en W3Schools: <https://www.w3schools.com/>
-  Pug: <https://pugjs.org/api/getting-started.html>
-  jQuery: <https://jquery.com/>
-  jQuery W3Schools: <https://www.w3schools.com/jquery/>
-  Bootstrap: <https://getbootstrap.com/>
-  Bootstrap3 W3Schools: <https://www.w3schools.com/bootstrap/>
-  GitHub: <https://github.com/>
-  GitKraken: <https://www.gitkraken.com/>
-  Visual Studio Code: <https://code.visualstudio.com/>