

**Planificador Web de Comidas Semanales**

* **MealWeek -**

# Ciclo: Desarrollo de Aplicaciones Web

# Curso: 2022/2023

# tutora: María Del Carmen Buenestado Fernández

# autor: Pedro Jesús Cruces Almendro

# Abstract

El proyecto “MealWeek” consiste en la realización de un sitio web con el que poder administrar y planificar todos los almuerzos y cenas que se realizan durante una semana. Se expondrán todos los requisitos previos necesarios para su desarrollo, así como detalles relevantes que tengas que ver con su proceso de creación. Se detallará todo lo relevante a los tipos de usuarios y las distintas funcionalidades que poseerán, el desarrollo de la base de datos, el contenido de los registros de cada tabla, los estilos que se usarán, así como el desarrollo de la API y todo el uso de tecnologías necesarias para la construcción de la web.

The “MealWeek” project consists in the creation of a website to manage and plan all lunches and dinners that take place during a week. All the necessary prerequisites for its development will be presented, as well as relevant details that have to do with its creation process. Everything relevant to the types of users and the different functionalities they will have, the development of the database, the content of the records of each table, the styles that will be used, as well as the development of the API and all the use of technologies necessary for the construction of the web, will be detailed.

**ÍNDICE**

[1. Justificación 5](#_Toc133333592)

[2. Introducción 6](#_Toc133333593)

[2.1 Presentación 6](#_Toc133333594)

[2.2 Medios Hardware y software a utilizar 7](#_Toc133333595)

[2.2.1 Hardware 7](#_Toc133333596)

[2.2.2 Software 8](#_Toc133333597)

[2.3 Planificación y temporalización 9](#_Toc133333598)

[2.3.1 Etapa 1 – Diseño de la web y de la Base de Datos 9](#_Toc133333599)

[2.3.2 Etapa 2 – Diseño de la API y escribir el código de la primera vista de la web 10](#_Toc133333600)

[2.3.3 Etapa 3 – Escribir el código de interacción entre la vista y la Base de datos 10](#_Toc133333601)

[2.3.4 Etapa 4 – Escribir el resto de vistas y restringir según tipo de usuario 10](#_Toc133333602)

[2.3.5 Etapa 5 – Mejorar la maquetación de la web y de las funcionalidades de cada usuario 10](#_Toc133333603)

[3. Objetivos 11](#_Toc133333604)

[4. Desarrollo 12](#_Toc133333605)

[4.1 Primeros pasos 12](#_Toc133333606)

[4.2 Configuración de archivos e IDE 13](#_Toc133333607)

[4.2.1 Organización por carpetas y archivos 13](#_Toc133333608)

[4.3 Base de datos 16](#_Toc133333609)

[4.4 API 18](#_Toc133333610)

[4.4.1 Ventajas y Desventajas de la arquitectura REST 18](#_Toc133333611)

[4.4.2 Dependencias necesarias 18](#_Toc133333612)

[4.4.3 Endpoints 18](#_Toc133333613)

[4.5 Tipos de usuarios y nivel de acceso 19](#_Toc133333614)

[4.6 Interfaces de usuario 19](#_Toc133333615)

[4.7 Requerimientos generales y específicos 19](#_Toc133333616)

[4.8 Tests 19](#_Toc133333617)

[5. Conclusiones 20](#_Toc133333618)

[Funcionalidades a futuro 20](#_Toc133333619)

[6. Bibliografía prevista para el desarrollo del proyecto 21](#_Toc133333620)

[Anexo I 22](#_Toc133333621)

[Gráfico 1 22](#_Toc133333622)

[Anexo II 23](#_Toc133333623)

[Boceto 1 23](#_Toc133333624)

## Justificación

La creación de un proyecto con estas características, surgen de la necesidad propia del autor para poder gestionar de una forma rápida y sencilla tanto las comidas que suceden a lo largo de una semana en una casa, como la compra que debe de realizarse para el no malgasto de alimentos. Además, todo ello también beneficia a cualquier familia por el ahorro económico de hacer una compra más controlada y ajustada.

La no existencia de herramientas similares que lleguen a todo usuario de una forma simple y directa, hacen que este proyecto sea aún más necesario y que todo el público en general sea objetivo como usuario de la web. Es por ello que una de las claves del proyecto debe residir en su simpleza, tanto en el estilo como en funcionamiento de cara al usuario.

## Introducción

### Presentación

El problema que se plantea en el proyecto es la realización de un portal que permita a cualquier usuario, planificar una semana de comidas y cenas, que generará, además, una lista de la compra que variará en función de lo que el usuario haya seleccionado. También se le dará la opción al usuario de almacenar los *plannings* que genere para que pueda recuperarlos las veces que lo desee.

Por lo que nuestra solución partirá por la diferenciación de dos tipos de usuarios: registrados y no registrados. Los usuarios no registrados solo tendrán acceso a la página principal en la que recibirán información acerca del proyecto y todas las funcionalidades que le puede aportar, mientras que los usuarios registrados podrán acceder a la toda la funcionalidad del sitio web.

Para satisfacer todas las funciones de los usuarios registrados, deberemos de contemplar al menos cinco componentes que debe tener nuestra aplicación web:

1. ***Planing Board***: en este componente, el usuario puede ver su planificación semanal y la lista de la compra resultante. Se le facilitará las opciones de editar, con el que podrá editar la planificación, y descargar como PDF.
2. **Editor del *Planning***: desde el editor, podrá construir el *planning* semanal a su gusto con comidas provenientes de su propia colección de comidas favoritas También podrá almacenar el *planning*, en caso de que le haya gustado, para poder recuperarlo cuando desee y/o modificarlo si así lo quisiera.
3. **Mis Plannings**: sección en donde se guardarán todos los *plannings* que quiera el usuario y donde podrá recuperarlos mediante un botón que lo enviará directo al editor.
4. **Página de comida**: cada comida tendrá su propia página en la que tendrá todos los datos que la identifican además de tener una descripción de la misma y los ingredientes necesarios para su preparación.
5. **Editor de comida**: En el editor de comida, se podrá crear comidas nuevas o editar las que el usuario haya creado. Además, si el usuario no encuentra los ingredientes que necesita, podrá crearlos y añadirlos a la base de datos general.

Además, habría que tener formularios de registro para los usuarios no registrados y de log in para el que ya esté registrado. También habría que contar con los formularios de creación de comidas y de ingredientes.

Con respecto a la API, deberemos de poder escribir y acceder a las tablas de la base de datos y luego restringir los usos según la funcionalidad de la web. Los datos para la conexión a la base de datos deberán de estar protegidos en un archivo de variables de entorno “env”, que procuraremos de incluir en el “gitignore”, archivo que nos dispone git para evitar que se filtren secretos con la subida al repositorio de “Github”.

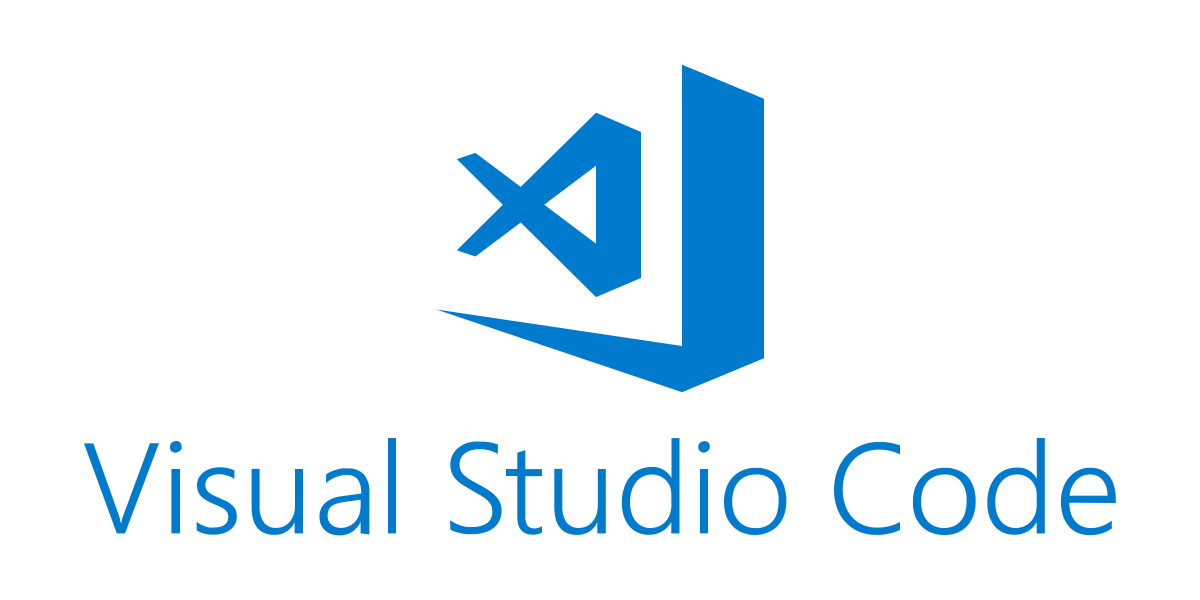
### Medios Hardware y software a utilizar

### Hardware

El hardware necesitado para la creación del proyecto no será especialmente diferente al usado para la creación del cualquier otro sitio web. En concreto, el ordenador utilizado para el desarrollo del proyecto es:

* Procesador: Ryzen 5 3600. 6 núcleos 12 hilos.
* Memoria RAM: 2 módulos de 8Gb Crucial Ballistix
* Disipador: Disipador AMD de serie con el procesador
* Placa base: B550 Aorus Elite V2
* Almacenamiento: Western Digital Blue SN550 SSD 1Tb con interfaz NVMe M.2
* Fuente de alimentación: Tacens Radix VII AG 600W
* Tarjeta gráfica: KFA2 NVIDIA GeForce RTX 3060 12Gb
* Monitor: AOC Gaming 24G2U 24" LED IPS FullHD 144Hz FreeSync
* Ratón: Logitech G502 Hero
* Teclado: Mars Gaming MK220

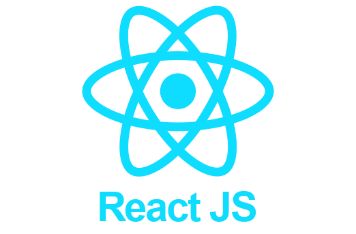
### Software

En cuanto al software, deberemos de utilizar un IDE para la escritura y control del código como es “Visual Studio Code”. En este IDE, podremos instalar diferentes extensiones que nos ayudarán en el progreso de desarrollo del proyecto y nos asegurará la correcta aplicación del código que vayamos escribiendo.

Necesitaremos de un control de versiones y documentación con el que poder llevar un control del software que vayamos produciendo. Para ello usaremos “Git” con el que podremos mantener este control mediante un repositorio en su sitio web “Github” y que podremos conectar fácilmente con nuestro IDE “Visual Studio Code”. Así desde el propio IDE, podremos realizar los *commit* que irán actualizando nuestro proyecto y podremos ver todos los cambios que han ido sucediendo de versión en versión.

Para el diseño de bocetos de la web y de las posibles vistas que pueden ser necesarias, usaremos “Adobe XD”. Con dicho software, podremos diseñar fácilmente cualquier vista que necesitemos, además de poder simular las interacciones entre páginas dentro del sitio web y la exportación de algunos elementos a código.

Para la edición y retoque gráfico de imágenes, logos y diferentes iconos, haremos uso de “Adobe Photoshop” y de “Adobe Illustrator”. Con el primero podremos modificar perfectamente cualquier imagen y adecuarla al formato web, y con el segundo, podremos modificar cualquier logo e icono que vayamos a utilizar, mediante vectores de una forma sencilla.

Respecto a las tecnologías a usar en el código, nos decantaremos por el *framework* *React*, ya que nos permitirá crear una página web dinámica mediante componentes que serán fácilmente mantenibles y actualizables. También usaremos CSS *Flex* y *Grid* para maquetar el sitio web, y varias dependencias que nos ayudarán con la escritura del código como “axios” o “Standard”.

### Planificación y temporalización

El proyecto se dividirá en diversas etapas que facilitarán la realización del mismo. Además de las etapas que se describirán a continuación, se puede consultar el gráfico 1 del anexo 1, en donde se plantea una temporalización aproximada por semanas de las diferentes etapas que componen el proyecto y su duración.

### Etapa 1 – Diseño de la web y de la Base de Datos

Comprendería el diseño gráfico de todos los elementos y vistas que componen el sitio web, además del diseño de la base de datos con todas las tablas y campos necesarios para el correcto funcionamiento de la web.

### Etapa 2 – Diseño de la API y escribir el código de la primera vista de la web

Esta etapa comprenderá la escritura del código necesario para la creación de la API que se conectará con la base de datos y con la que se accederán a todos sus datos. También se procederá a escribir el código de la primera vista de la web que funcionará como página de inicio para el usuario registrado.

### Etapa 3 – Escribir el código de interacción entre la vista y la Base de datos

En esta etapa, ya dispondremos tanto de la API como de una vista básica de la web con la que podremos comenzar a escribir todo el código, que relacionará la web con la base de datos y con las que podremos cumplir con los principios CRUD (*Create Read Update Delete*) desde el sitio web.

### Etapa 4 – Escribir el resto de vistas y restringir según tipo de usuario

Una vez que ya disponemos de toda la funcionalidad de la base de datos y podemos controlarla desde la web, procedemos a realizar el resto de vistas necesarias para los distintos usuarios y asegurarnos de restringir todos los permisos de cada usuario a su nivel correspondiente.

### Etapa 5 – Mejorar la maquetación de la web y de las funcionalidades de cada usuario

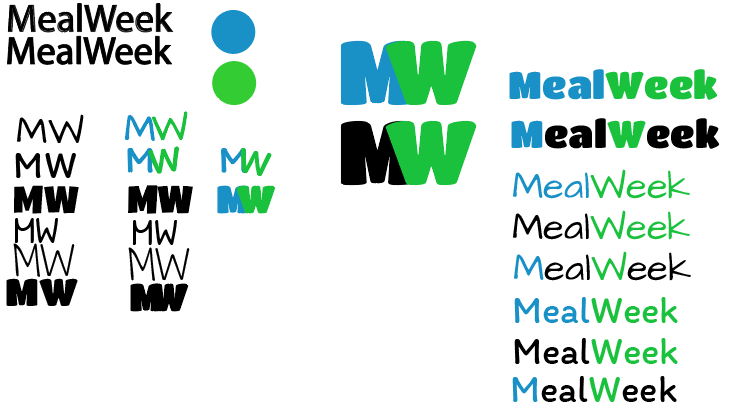
En esta etapa se ultiman los detalles finales relacionados tanto con el diseño como con la funcionalidad para que el usuario final no encuentre ningún tipo de problema, y finalizamos pasando la web a producción.

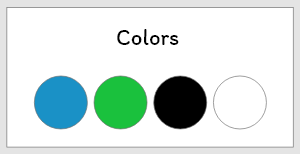
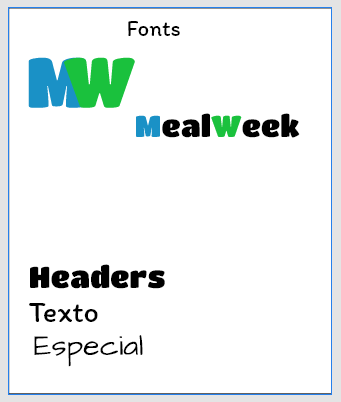
## Objetivos

* Creación de un sitio web que permita la realización de una planificación semanal de almuerzos y cenas y que genere una lista de la compra en base a dicha planificación
* Permitir que el usuario pueda elegir rellenar la planificación de forma manual, con las comidas que le gusten.
* Permitir que cada usuario disponga de su propia galería de comidas donde podrá guardar las que más le gusten.
* Permitir que cada usuario disponga de una sección donde podrá consultar *plannings* que le haya gustado y que desee repetir.
* Permitir que cada usuario pueda crear comidas nuevas que nutrirán la colección general además de la suya propia.

## Desarrollo

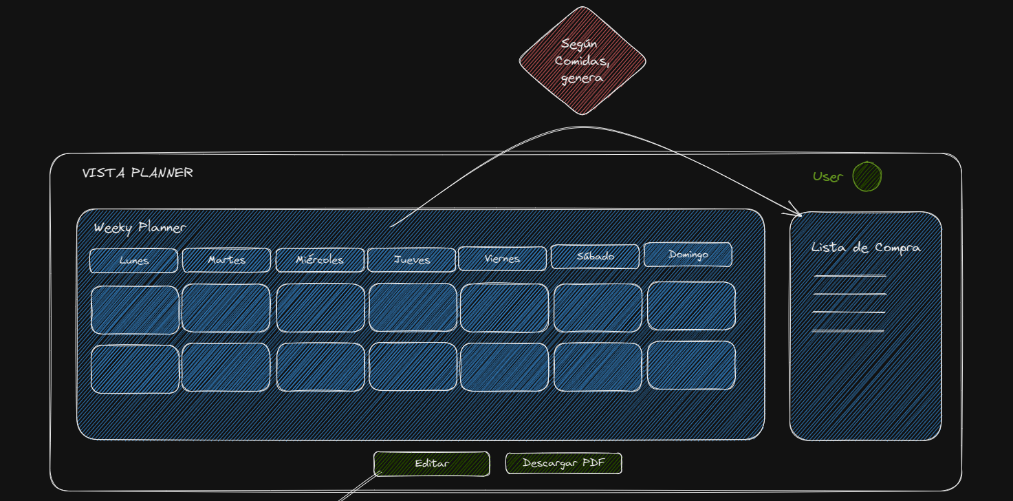
### Primeros pasos

Para asentar buenas bases a los cimientos del proyecto, deberemos de plantear en un comienzo la guía de estilo que usaremos en el mismo y algunos bocetos sobre las vistas más importantes.

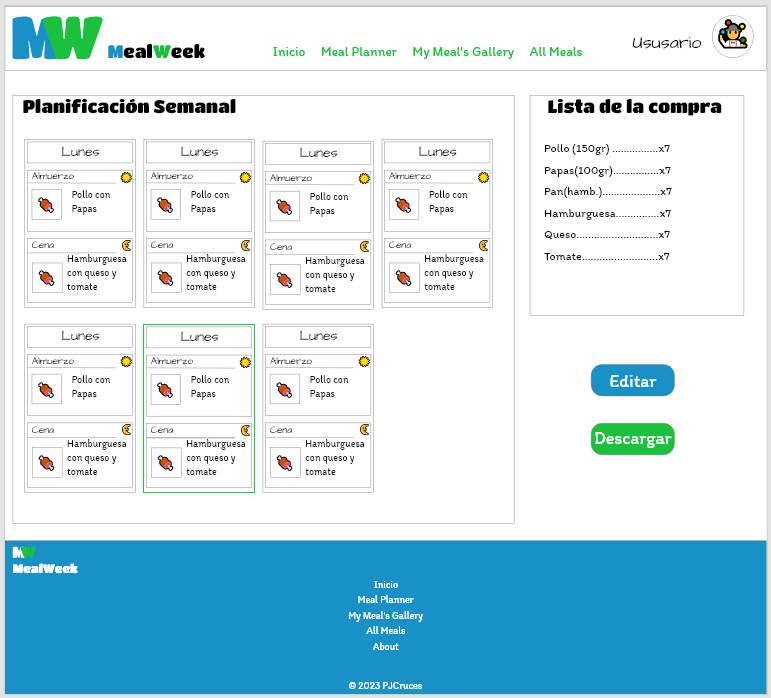
Para el logotipo, se utiliza la fuente “Titan One” y se mezclan los glifos de la M y la W, siglas del título que da nombre al proyecto. Además, se utilizarán como colores claves el azul *#1A91C6* y el verde *#1AC13D*, combinándolos con blanco y negro dependiendo de la sección de la web para el fondo o para el color de la fuente. También usaremos pequeñas variaciones de estos para el resaltado de enlaces y botones.

Continuando con el diseño, para los *headers* usaremos la misma fuente del logo, para el texto general usaremos la fuente “Itim” y para algunos textos especiales, como los días de la semana de la planificación, usaremos la fuente “Architects Daughter”.

Lo mejor para la instalación de las fuentes es convertir exclusivamente las variantes que nos interese al formato web WOFF2. Este formato, es la opción más óptima para cargar las fuentes a nuestro sitio web sin que tengamos que sufrir grandes tiempos de espera y sin tener que depender que el usuario disponga de las fuentes elegidas ya instaladas en su equipo o dispositivo.

Respecto a los bocetos iniciales, podemos usar la herramienta “Excalidraw.com” con la que podremos dibujar fácilmente un pequeño esquema sobre la vista de la web y el cómo se comportará su interactividad. En este boceto (que se puede consultar en el boceto 1 del anexo 2), se puede observar cómo habría dos vistas principales, y el resto de ventanas podrían ser anexadas a estas vistas, usando *modals* o diálogos; o podrían configurarse en alguna vista nueva.

Con el boceto realizado, pasamos a finalizar la vista preliminar del estilo con el software “Adobe XD”. En esta herramienta, podremos realizar un boceto lo más cercano al resultado final en lo que a estilo se refiere. Más tarde podremos usarlo como referencia para poder realizar una maquetación del sitio web lo más perfecta posible y asegurando que el sitio se realiza de forma *responsive*, es decir, que no importa la pantalla o el dispositivo en el que se visualice, todos sus contenidos se verán de la manera correcta.

Una vez que el diseño de la web y que los elementos que lo componen están claros sobre la guía de estilo, estaremos listos para proceder a comenzar con la realización del proyecto.

### Configuración de archivos e IDE

Una vez que tenemos la guía de estilo planteada, necesitamos establecer que tipos de tecnologías vamos a usar para la construcción de todo el sitio web. Teniendo en cuenta que vamos a utilizar tecnologías de *front-end* para todo el control del diseño y del cliente, y de *back-end* para la creación y gestión de la base de datos, una de las soluciones más adecuadas sería “Next.JS”.

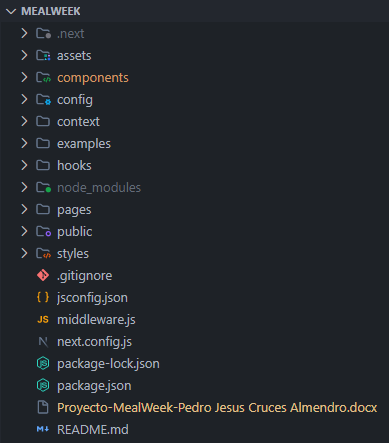
“Next.JS” es un *framework* de *React* (que a su vez es un *framework* de *JavaScript*), con el que podremos construir un sitio web con todas las ventajas que nos ofrece de por sí *React*, sumado a la posibilidad de crear componentes que se rendericen en el servidor (*ServerSideRendering*), y la posibilidad de crear fácilmente un enrutado de páginas y *endpoints* para una API. Además, dispone de la plataforma “Vercel” donde se pueden desplegar aplicaciones construidas con este *framework* de forma gratuita.

La instalación del *framework* es sencilla, pero nos creará archivos y carpetas por defecto que deberemos de entender para su buen uso, ya que, de lo contrario, provocaremos muchos errores que nos entorpecerá la progresión del proyecto.

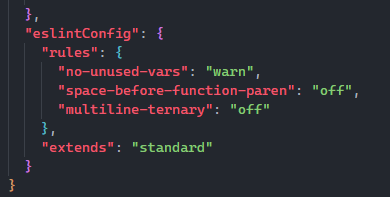
Para proceder solo basta con entrar al terminal de nuestro ordenador y escribir el comando:  
“npx create-next-app@latest”

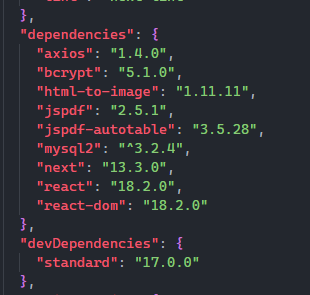
La propia aplicación te guiará con la configuración inicial.

### Organización por carpetas y archivos

Respetar la organización de directorios en Next.JS es crucial para su funcionamiento. Dentro de la carpeta de nuestro proyecto, donde hemos instalado el *framework*, encontraremos los directorios creados por defecto, además de algunos que crearemos para una mejor organización de nuestros archivos:

* **.*next***: dónde se guardará información y archivos de configuración relevantes para el buen funcionamiento del *framework*.
* **Assets**: carpeta en la que se almacenará todo el contenido extra del que necesite la web, como imágenes, fuentes, SVG, etc …
* **Components**: en esta carpeta se almacenarán todos los componentes que crearemos y que iremos reutilizando en las diferentes páginas de la web.
* **Config**: en esta carpeta, guardaremos archivos con extensión “.js” que nos servirán para poder configurar algún elemento de nuestro sitio web, como el componente que servirá para poder conectarnos a la base de datos.
* **Context**: en esta carpeta se almacenan los contextos. Un contexto se trata de un componente global que alberga información que puede pasar a otros componentes del sitio web. Con estos componentes podemos disponer de estados globales con los que compartir entre componentes de una forma sencilla.
* **Examples**: esta carpeta será para albergar en ella archivos JSON. Estos archivos estarán creados a propósito a imagen del resultado que querremos reproducir tras llamar a un *endpoint* de la API. Gracias a estos archivos, podemos escribir toda la funcionalidad de la web sin depender de escribir nuestra API primero y poder observar el comportamiento de los datos desde el primer momento.
* **Hooks**: en esta carpeta podremos almacenar los *custom hooks* que vayamos refactorizando del código y que nos permitirá reutilizar, no solo en este mismo proyecto, sino en cualquier otro por su especial construcción.
* ***node\_modules***: directorio en el que se instalan todos los módulos necesarios para el funcionamiento de la máquina virtual de “node” que ejecutará nuestro proyecto en local y con el que podremos crear una vista previa de nuestro proyecto mientras lo escribimos. También nos genera una compilación de nuestro código cuando lleguemos a la conclusión del mismo.
* ***Pages***: este directorio es de los más importantes. En su interior, en el que cualquier directorio o archivo .js o .jsx (o sus variantes con *typescript*), serán considerados como páginas y rutas independientes, por lo que no habrá que configurar un *router* como sí podríamos hacer en *React*. Además, en su interior, tenemos el directorio llamado API, en el que al igual que sucede con el directorio “pages”, cualquier archivo o directorio en su interior será considerado como una ruta, pero esta vez hacia una API. Por lo que la creación y manejo de una API en este *framework* se vuelve muy fácil y conveniente.
* ***Public***: este directorio no esconde muchos secretos, ya que su función es exactamente la misma que en cualquier otro proyecto web.
* ***Styles***: *Next.JS* usa por defecto la tecnología *CSS Modules*, lo que permite separar el archivo CSS en módulos distintos según los componentes de nuestra web o páginas, creando así un rango más específico. Por ejemplo, podríamos escribir los mismos nombres de clases en distintos módulos CSS y estos, no entrarían en conflicto en ningún momento. Este directorio, por tanto, es el utilizado para albergar todos los módulos, además de incluir un globals.css que afectará a todas las páginas en general.
* ***.env***: este archivo es muy importante que se encuentre en la lista de archivos ignorados a la hora de subir al repositorio de “Github” (se explica a continuación). Esto se debe a que se trata de un archivo solo con variables que se utilizan en entorno de desarrollo. Las variables que se albergan en dicho archivo son de carácter secreto, como las credenciales de acceso a la base de datos, las URL y las KEY necesarias para acceder a APIs externas, etc…
* .***gitignore***: este archivo es uno de los más importantes para la no filtración de datos sensibles. Al usar *Git* como nuestro controlador de versiones, necesita saber que archivos debe o no debe de incluir en el repositorio que estará disponible en la red. Es por ello, que, dentro de este archivo, deberemos de incluir todas las rutas (relativas) que no deberán de subirse al repositorio de “Github”.
* ***jsconfig.json***: este archivo, configura nuestro editor y nos permite incluir o excluir directorios de cara a la compilación.
* ***Middleware*.js**: este archivo es el que reproduce “Next.js” cuando trata de abrir alguna página de nuestro sitio. Como su nombre indica, sirve como intermediario entre las comunicaciones y nos puede permitir, entre otras cosas, leer el token donde se almacenará la información del usuario. Si este no existiera, impediría la entrada a las páginas restringidas solo para los usuarios registrados, y si el token fuera válido, daría paso a la página que desea visitar el usuario.
* ***next.config.js***: archivo en el que incluiremos configuración específica de “Next.JS” y en el que, por ejemplo, deberemos de incluir dominios para poder usar los componentes *Image* si para estos utilizamos enlaces externos a nuestro sitio.
* ***package.json***: en este archivo de formato JSON, aparece la información relativa a nuestro proyecto, como el nombre y la versión, además de comandos, como “npm run dev” con el que tendremos una renderización en vivo de nuestro proyecto a nivel local para poder disponer de una vista previa, o “npm run build” con el que podremos compilar y llevarlo a producción. Si desplegamos en “Vercel”, este último paso no sería necesario por la posibilidad que te ofrece de poder importar tu proyecto directamente desde el repositorio de “Github” y de realizar una compilación directa. También aparece en detalle, las dependencias necesarias para el funcionamiento de la web y la versión requerida. Además, podremos incluir instrucciones respecto a la configuración del *linter*, con la que ayudarnos a la hora de escribir código.
* ***package-lock.json***: Funciona como una copia de las versiones de las dependencias definidas en el *package.json*. Cualquier cambio que hagamos en el *package.json*, al ejecutar la actualización de una dependencia o al instalarlas todas con “npm install”, se actualizará *package-lock.json* con la versión más reciente instalada.
* ***README.md***: este archivo, como en el resto de proyectos web, funciona como la documentación principal de nuestro proyecto. Que, en nuestro caso, este mismo documento podría realizar la misma función.

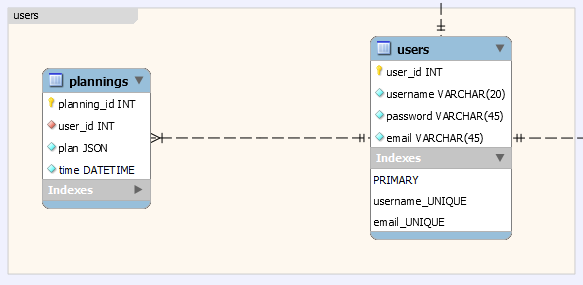
Una vez que disponemos y entendemos la jerarquía de los directorios y su función, podremos pasar a configurar el IDE. Para este proyecto, usaremos el *software* *VSCode*, IDE que además de facilitarnos la conexión con nuestro repositorio de *Github*, también nos permite la instalación de diferentes *plugins* que ayudarán con la depuración del código. Entre otros, usaremos un plugin para desplegar un *intellisense* sobre *javascript* y *react*, y un paquete que nos configurará el *linter*. En primer lugar, el *intellisense* nos sugerirá a medida que vamos escribiendo, métodos, clases y atributos que pertenezcan a lo que estemos escribiendo, además de variables, constantes y funciones que tengamos en nuestro documento. Por ejemplo, si escribimos el objeto “window”, al poner el punto, se abrirá un desplegable que te sugerirá todos los métodos accesibles de dicho objeto.

Respecto al *linter*, es la herramienta que nos señaliza la sintaxis que no está escrita de manera correcta, o si hay cualquier otro error que hubiese en el código. Este *linter*, se apoya de una configuración que nos permite personalizar a nuestro gusto puesto que hay reglas en sintaxis que corresponden más a un estilo de escritura, que, al correcto funcionamiento de nuestro código, como puede ser el escribir “;” después de cada instrucción en *JavaScript*. Para este caso, instalaremos la dependencia “standard” con el que obtendremos una configuración del *linter* casi perfecta. Para completarla, solo deberemos de añadir las reglas que queramos cambiar al *package.json*, bajo *eslintConfig*, y asignarle un valor *off*, *warn* o *error*, según como queramos que nos lo señalice.

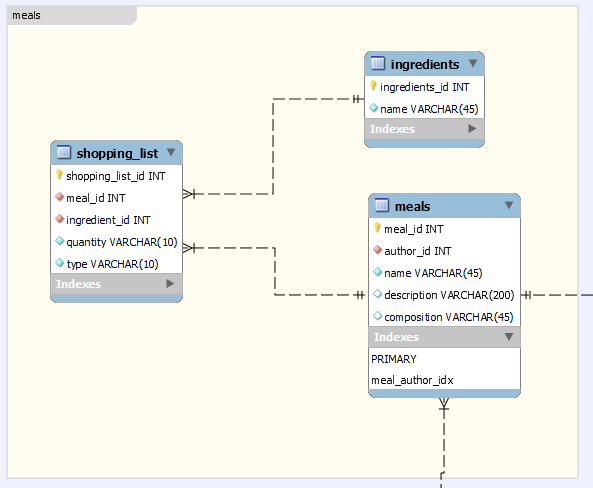
Además, necesitaremos de la instalación de dependencias que nos faciliten, no solo la escritura de código, sino la habilitación de ciertas capacidades a nuestra aplicación web. De hecho, necesitaremos de la instalación de la dependencia “mysql2” con la que podremos establecer conexión con nuestra base de datos y realizar cualquier tipo de *query*, y la dependencia “bcrypt” con la que podremos encriptar *passwords* y realizar su posterior autenticación. Además, para recuperar estos datos, podremos utilizar la función “fetch” y configurar los parámetros que recibe la función o también podemos utilizar la dependencia “axios” con la que podremos ejecutar nuestras consultas de una forma muy sencilla. Todo ello se explicará con mayor profundidad en las próximas secciones.

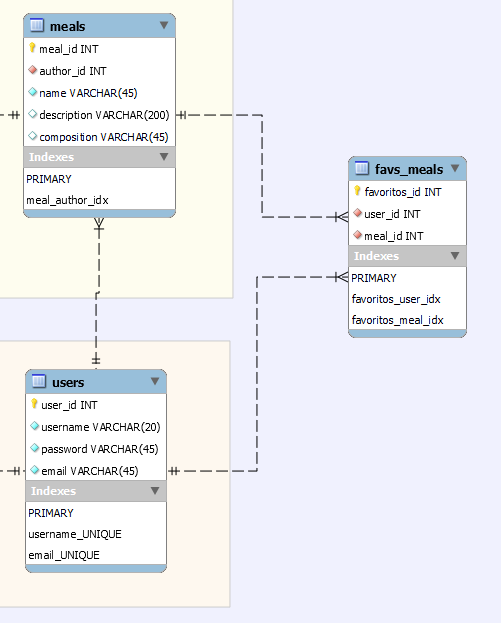
Teniendo ya una buena configuración del IDE, podemos comenzar a escribir la solución para nuestro proyecto.

### Base de datos

Para comenzar, nuestra solución deberemos de diseñar una base de datos en la que almacenaremos todos los datos de autenticación de usuarios, comidas y las comidas favoritas que tenga cada usuario. Usaremos MySQL, que se trata de una base de datos con la que podremos, tanto almacenar como consultar datos de forma estricta para poder garantizar la integridad de los mismos. Dentro de la base de datos, el esquema (que se puede consultar en su totalidad en el Anexo 1, Diagrama 2), deberá de estar formado por tres tablas:

* ***Users***: Esta tabla contendrá los datos de cada usuario que se registre en el sitio. Contendrá los campos de *id*, *username*, *password* y *email*, de los cuales id, *username* y *email*, deberán de ser únicos para garantizar la correcta identificación de cada usuario. También hay que considerar de que no es buena práctica almacenar *passwords* en la base de datos sin una correcta gestión de ellos. Es por ello, que necesitaremos de algunas herramientas que nos ayuden a su encriptación y correcto almacenaje (se explicará en la siguiente sección de forma más detallada.)
* **Plannings**: Esta tabla contendrá los *plannings* guardados de cada usuario. La forman los campos de *planning\_id*, *user\_id*, *plan* y *time*. El campo de *planning\_id* y servirá para poder identificar cada plan en las consultas. El campo de *user\_id* será necesario para saber a quién le pertenece el *planning*, y *plan* será en donde se guarde el *planning* en formato JSON.



* **Meals**: Esta tabla contendrá los datos de cada comida creada por los usuarios y que estará disponible para todos ellos. Contendrá los campos de *id*, *author*, *name*, *description*, *composition* e *ingredients*, de los cuales *id* y *name*, deberán de ser únicos para asegurar que no se almacenan dos entradas iguales. El campo de *author*, nos ayudará a relacionar esta tabla con la de *users*, y el campo de *composition*, será el que nos ayude a identificar la composición de las comidas y así evitar posibles alergias o comidas vegetarianas, etc…
* **Ingredients**:Esta tabla es necesaria para el control de los ingredientes que se añaden a las comidas. Sólo contendrá dos campos: *ingredients\_id* y *name*. Cabe añadir, que el campo *name* deberá ser único para que no exista conflictos de duplicados.
* **Shopping\_List**:Esta tabla es la que especifica que cantidad de ingredientes componen las comidas. De esta manera, podemos generar la lista de la compra de forma adecuada. Estará formada por *shopping\_list\_id*, *meal\_id*, *ingredient\_id*, *quantity* y *type*. Necesitamos de los *id* de las comidas y los ingredientes para su correcta relación y, además, registraremos la cantidad y el tipo de medida usada, como “g” o “ml”.
* **Favs-Meals**: En esta tabla se almacenarán los datos de las comidas que los usuarios seleccionen como favoritas, y, por tanto, aparecerán en un apartado donde tendrán todas las comidas favoritas. Solo contendrá los campos de *id*, *user\_id* y *meal\_id*, ya que sólo necesitaremos identificar que usuario ha seleccionado como favorita que comida.

### API

Para el almacenaje y la consulta de datos necesitaremos de una API. Esta API se conectará directamente con la base de datos y nos permitirá acceder a todos los registros que necesitemos. Para su correcto uso y para asegurar que estamos creando una aplicación usando buenas prácticas, tendremos que tener en cuenta los principios de la arquitectura REST:

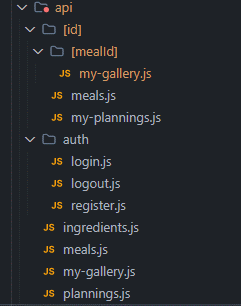
* Debe seguir una arquitectura cliente-servidor con solicitudes gestionadas mediante HTTP.
* Comunicación entre el cliente y el servidor sin estado.
* Datos que pueden cachearse para optimizar las interacciones entre servidor y cliente.
* Interfaz uniforme entre los elementos
* Sistema de capas que organiza en jerarquías los servidores que participan en la respuesta al cliente

### Dependencias necesarias

* Mysql2: con esta dependencia, podremos realizar las consultas SQL desde la API. Esta dependencia necesita de una inicialización con la configuración de acceso a la base de datos. Estos datos son muy importantes y por tanto deberemos almacenarlos en un archivo de variables de entorno, un archivo que solo funcionará de forma local y con el que nos evitaremos tener que exponer nuestras claves de acceso, librándonos así de cualquier ataque malicioso o uso inadecuado.
* Axios: con esta dependencia podremos realizar las peticiones a la API desde el lado del cliente o del servidor. Su uso es muy sencillo y muy simple con el que evitaremos los errores de generar una mala petición.
* Bcrypt: con esta dependencia podremos encriptar los *passwords* que los usuarios nos faciliten para su identificación. Por motivos obvios, almacenar estos *passwords* sin una protección adecuada, repercutiría negativamente sobre estos datos y los dejaría desprotegidos. Además de encriptar, “Bcrypt” también te permite verificar si los datos se corresponden con una cadena de *string* que se le facilite. Si fue la misma que generó la encriptación, dará un true como resultado, sino dará un false. De esta manera, la dependencia solo encripta la información y más tarde solo la verifica, sin necesidad de desencriptar, lo que nos asegura una protección óptima para nuestros datos.
* Jwt: esta dependencia se encargará de generarnos un JSON web token, es decir un objeto en el que podremos almacenar datos cifrados. Para ello, en los parámetros de creación del token se le debe de acompañar de una clave con la que se cifrará y con la que también verificará que el token es válido. Esta clave, debe de ser almacenada en el archivo de variables de entorno para evitar su filtración.
* Cookie: con esta dependencia podremos crear una cookie en la que podremos almacenar el JWT en el lado del cliente. Con esto, podremos validar si el usuario ha hecho *log in* o no y poder consultar algunos datos que almacenemos, como por ejemplo las comidas favoritas que tengan cada usuario.
* Jose: realiza la misma función que JWT, con la salvedad de que JWT no funciona en los middlewares, mientras que con “Jose” no habría ningún problema. Esto nos va a permitir restringir la entrada del usuario no registrado en las páginas y subpáginas mediante la comprobación en el middleware de si el token guardado en la cookie es válido o no.

### Endpoints

Para que nuestra API funcione como debiera, necesitaremos de diversas rutas o *endpoints* con los que, mediante las diferentes peticiones, podremos acceder a la base de datos y o bien, insertar datos en ella o bien recuperarlos.



* /[id]: Next.JS permite que se creen parámetros en las peticiones mediante directorios y colocando en su nombre una variable entre corchetes. De esta manera, podremos acceder al valor de dicha *query* desde el objeto *Request*. En esta ruta se recibe en concreto un número que corresponderá con un “ID”.
* /[id]/[mealId]/myGallery: en esta ruta, se reciben dos parámetros y permite borrar una comida mediante su ID y la del usuario que desee eliminarla.
* [id]/meals: en esta ruta se puede consultar toda la información de una comida en concreto mediante su Id, que es recibida como parámetro en la URL.
* [id]/my-plannings: en esta ruta se puede eliminar un plan en concreto en función del id recibido como parámetro.
* /auth/login: en esta ruta se puede validar un usuario y crear el JWT que se almacenará en las cookies.
* /auth/logout: en esta ruta se valida que exista un token válido para después eliminarlo y limpiar las cookies del cliente.
* /auth/register: en esta ruta recibe el formulario de registro y lo inserta en la base de datos. La contraseña utilizada se encripta con *bcrypt*.
* /ingredients: en esta ruta podremos tanto crear ingredientes como consultarlos para poder poner a disposición del usuario, la lista de ingredientes para el formulario de creación de comidas.
* /meals: en esta ruta podremos tanto crear comidas como consultarlas para poder recibir todos los datos que necesitemos de ellas.
* /my-gallery: en esta ruta podremos obtener las comidas favoritas de el usuario mediante el paso del id en el cuerpo de la petición y la creación de comidas favoritas para un usuario en concreto.
* /plannings: en esta ruta podremos obtener los *plannings* que haya guardado el usuario y también podremos registrar en la base de datos los nuevos *plannings*.

### Interfaces de usuario

Con el lado del servidor definido, podemos definir las interfaces que experimentarán los usuarios.

### Home o landing

### Formularios de registro y *log in*

### Vista de *planning*

### Vista del editor

### Vista de mis *plannings*

### Vista de todas las comidas

## Conclusiones

En conclusión, la experiencia de realizar un proyecto tan completo como este, en dónde hemos puesto a pruebas tanto nuestras habilidades como desarrollador *front-end*, como de *back-end* es muy positiva y nos permite poner nuestros conocimientos en práctica. Durante el proceso, nos hemos encontrado con diversas dificultades, sobre todo relacionado con la API y con su buen funcionamiento, pero se pudieron solventar todas las adversidades sin ningún problema.

En mi opinión, las tecnologías elegidas fueron las más idóneas ya que el disponer de un manejo de la API como el que plantea Next.js, facilitó en gran parte a obtener y enviar todos los datos tal y como se esperaba. El uso de componentes para el *front-end* también supuso una gran ventaja para la descomposición de las páginas y para la reutilización de estos mismos en este mismo proyecto o en otros distintos.

A pesar de todo, el trabajo aún no ha concluido y aunque la página sea funcionalmente correcta, hay muchas otras grandes ideas que se pueden implementar y hacer de “MealWeek” un proyecto aún mejor.

### Funcionalidades a futuro

* Guardar como PDF: Aunque el botón funciona y la funcionalidad de la dependencia de JsPDF nos permite obtener el PDF, el resultado que obtenemos de este no es el más óptimo y en algunos casos no es positivo. Una posible solución sea la creación de una tabla en la que se plantee la información del planning, en lugar de sacar una captura al elemento HTML puesto que los errores que obtenemos residen mayormente por culpa de esa captura.
* Perfil de usuario: Permitir que el usuario disponga una página personal donde poder realizar gestiones de cambio de datos o donde podría darse de baja.
* Botón random: Crear un botón que permita colocar en el planning comidas al azar, ya sea siguiendo un criterio o no. Y estos criterios podrían estar relacionados con la composición de las comidas o sus alérgenos.

## Bibliografía prevista para el desarrollo del proyecto

CSS-Tricks. (2023). *Home*. [Consultado el 06 de 04 de 2023] de CSS-Tricks: <https://css-tricks.com/>

Durán, M. Á. (2023). *Home*. [Consultado el 06 de 04 de 2023] de Midudev: <https://midu.dev/>

MDN. (2023). *Home*. [Consultado el 06 de 04 de 2023] de MDN Web Docs: <https://developer.mozilla.org/en-US/>

Meta Open Source. (2023). *Home*. [Consultado el 06 de 04 de 2023] de React: <https://react.dev/>

Next.JS. (2023). *Home*. [Consultado el 06 de 04 de 2023] de Next.JS: <https://nextjs.org/>

Oracle. (04 de 04 de 2023). *MySQL 8.0 Reference Manual*, revision: 75301. [Consultado el 06 de 04 de 2023] de MySQL: <https://dev.mysql.com/doc/refman/8.0/en/>

Powell, K. (2023). *Home*. [Consultado el 06 de 04 de 2023] de Kevin Powell: <https://www.kevinpowell.co/>

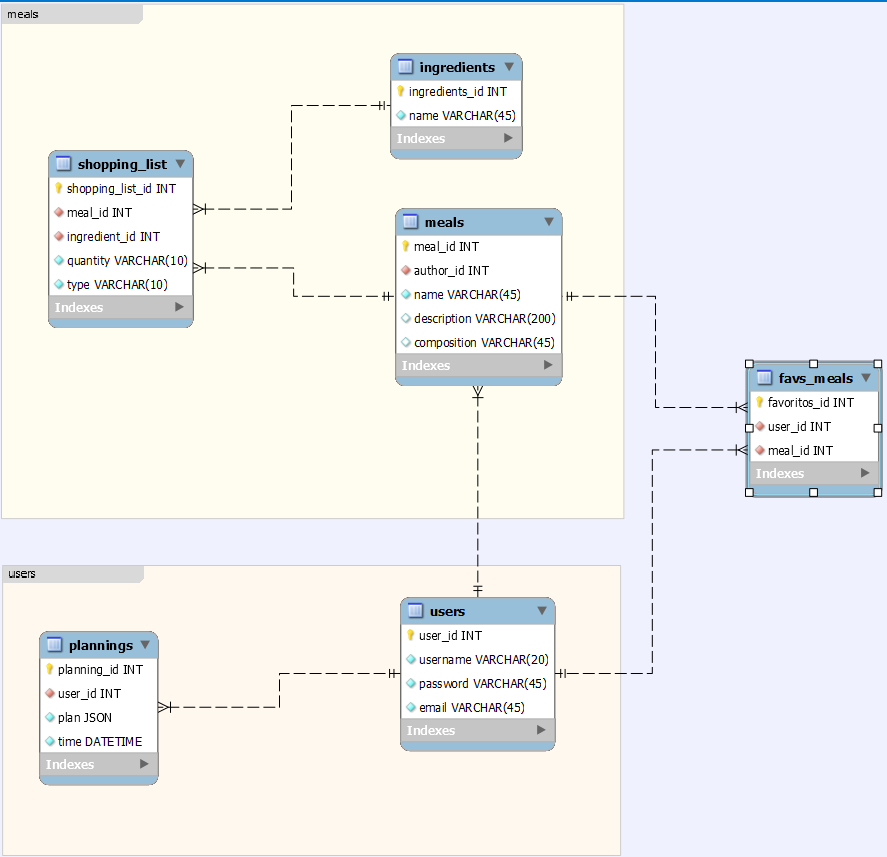
# ANEXOS

## Anexo I

### Gráfico 1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ETAPAS / SEMANAS** | **ABRIL** | | | | **MAYO** | | | |
| **1** | **2** | **3** | **4** | **1** | **2** | **3** | **4** |
| **Etapa 1 – Diseño de la web y de la Base de Datos** | x | x |  |  |  |  |  |  |
| **Etapa 2 – Diseño de la API y escribir el código de la primera vista de la web** |  | x | x |  |  |  |  |  |
| **Etapa 3 – Escribir el código de interacción entre la vista y la Base de datos** |  |  | x | x | x |  |  |  |
| **Etapa 4 – Escribir el resto de vistas y restringir según tipo de usuario** |  |  |  | x | x | x | x |  |
| **Etapa 5 – Mejorar la maquetación de la web y de las funcionalidades del usuario** |  |  |  |  |  |  | x | x |
| **Conclusión del proyecto** |  |  |  |  |  |  |  | x |

### Diagrama 2



## Anexo II

### Boceto 1

